信号保安装置への定理証明技術の適用に関する研究

寺田, 夏樹

https://doi.org/10.15017/1543999

信号保安装置への
定理証明技術の適用に関する研究

2015 年 8 月

寺 田　夏 樹

目次

図目次

表目次

略語一覧

API Application Programming Interface
ATC Automatic Train Control 自動列車制御装置
ATS Automatic Train Stop 自動列車停止装置
BART B Automatic Refinement Tool
BDD Binary Decision Diagram 二分決定木
BMC Bounded Model Checking 有界モデル検査
CASE Computer Aided Software Engineering
CD Code Disjoint
CORBA Common Object Request Broker Architecture
CPU Central Processing Unit
CSP Communicating Sequential Processes
CTC Centralized Traffic Control 中央列車制御
CTL Computation Tree Logic 計算木論理
DSP Digital Signal Processor
FS Fault Secure
GUI Graphical User Interface
HOL Higher Order Logic 高階論理
IC Integrated Circuit
IDE Integrated Development Environment 統合開発環境
IDL Interface Definition Language インタフェース記述言語
IEC International Electrotechnical Commission 国際電気標準会議
ISO International Organization for Standardization 国際標準化機構
JDK Java Development Kit
JIS Japan Industrial Standards 日本工業規格
LPF Logic of Partial Functions
LTL Linear Temporal Logic 線形時相論理
ME Microelectronics
ML Meta-Language
NASA National Aeronautics and Space Administration アメリカ航空宇宙局
NP Non-deterministic Polynominal time 非決定性多項式時間
ORB Object Request Broker

PRC Programmed Route Control 自動進路制御装置
PVS Prototype Verification System
Promela Process meta languge
RATP Régie Autonome des Transports Parisiens パリ交通公団
SAT Satisfiability problem 充足可能性問題
SFS Strongly Fault Secure
SMT Satisfiability Modulo Theories
SMV Symbolic Model Checker
TCC Type Check Condition
TD Train Detection
TSC Totally Self Checking
UML Unified Modeling Language
VDM Vienna Development Method
VDM-RT Vienna Development Method Real Time
VDM-SL Vienna Development Method Specification Language

第 1 章

はじめに

鉄道の安全な運行を支えているのが信号保安装置と呼ばれる装置群である．信号保安装置の機能としては，駅間において列車同士を正面衝突させないように，線路上を列車が走行する方向を制御する，あるいは後続の列車が先行する列車に追突しないように，後続の列車の速度を制御（制限）する，また駅構内において列車を正面衝突あるいは側面衝突させないように，駅構内のポイント（線路を 2 方向に振り分ける装置）や信号機を制御するなどといったものが挙げられる．

これらの機能を具体的に担う基本的な装置として，列車検知装置，閉そく装置，信号機，転てつ装置，連動装置，ATS （Automatic Train Stop : 自動列車停止装置），ATC（Automatic Train Control : 自動列車制御装置）がある．さらに，運行を円滑かつ効率よく行うための運行管理装置，踏切の制御を行う踏切保安装置を加えて信号保安装置が構成されている．

鉄道では駅間において閉そく（block）という概念を導入し，1 つの閉そくには 1 列車しか在線させないような制御を行っている．閉そくには軌道回路と呼ばれる列車検知装置が設備され，これにより列車を検知し，検知結果に従って各閉そくにおける安全な速度を決定する．この安全な速度に従って信号機を制御し，運転士に速度を指示する．これが閉そく装置と呼ばれるものである．さらに車両が停止すべき位置や定められた速度を超えないように車両にブレーキをかけるのが ATS や ATC といった装置である．

また駅構内においては，進路と呼ばれる列車の走行経路を定め，競合する（経路が重なる）進路が同時に走行可能とならないように，進路に対応した信号機を制御して，運転士に走行を指示したり，進路に関連するポイントを鎖錠（ロック）し，列車が進路を走行している途中でポイントが転換して列車が脱線しないようにしたりしている．このように駅構内の信号機やポイントを制御しているのが連動装置（interlocking）である．なお，転てつ装置とはポイントを動かしたり，鎖錠したりする装置をいう．

これら基本的な信号保安装置には高い安全性が求められる．最初に実現された装置には機械的なものが使われたが，やがてリレー（継電器）が実用化され，これが信号保安装置の大部分を占めるようになった．ここで重要なのは，リレーが故障時の状態が一方に固定されるという非対称な故障特性を有しているという点にあり，これにより，装置が故障しても安全側に制御されるように論理を組み立てることが可能となった．閉そく装置や連動装置はリレーで構築された．大規模な駅の連動装置はリレーでなければ実現できなかった．また ATS や ATC も論理を担う部分にはリレーが使用された．

1970 年代以降のコンピュータの発達により，連動装置を電子機器で実現する試みが始まったが，

ここで問題となったのは，電子機器はそのままでは非対称な故障特性が得られないという点である．そこで装置を多重化し，その不一致を検出した場合に装置を安全側に止めるという技術が開発された．不一致を検出する比較器については自身が故障しても安全側に制御可能なフェールセーフ性を有するものを使用する．こうして電子連動装置と呼ばれる装置が実現した．世界で初めての電子連動装置として，スウェーデン・ヨーテボリにて 1978 年に実用化第 1 号機が稼働を開始したが，日本でも 1985 年に最初の実用機が稼働を開始した．

そのほかに，1985 年の ATC 車上装置の ME（Microelectronics，電子）化や電子閉そく装置と呼ばれるシステムの実用化などもあり，このあたりから日本において電子化機器による信号保安装置が次々と導入されていく．その後，電子化機器のハードウェアに関してはプロセッサの高速化や大規模化にどう対応するかといった問題はあったが，安全性を確保するという視点で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についてはそれほど特別な技術は使われていない．例えばコーディング規則を縛ったり，プログラムをシングルスレッドとして一定周期で決められた処理ルーチンを起動する仕組みにするなど，安全性には配慮されているが，これらの技術はプログラムを誤らないように単純化するという考え方に基づいている．プログラムが正常でないことを検出する仕組みは備わっているものの，プログラムの正当性を確保する手段は依然としてレビューやテストによる所が大きい．

これに対し，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に関心を持っている．

フォーマルメソッドとは仕様の形式化を通じてプログラムやシステムの信頼性を向上させるための技術である．フォーマルメソッドには大きく分けて 2 つのアプローチがある．1 つは，形式化を通じて仕様のあいまいさを排除し，ラピッドプロトタイピングを可能とすることで仕様の品質を上げる Ligth Weight な考え方であり，もう 1 つは，仕様の自動検証や仕様から生成される条件の定理証明により仕様の誤りを見つけたり，仕様に誤りのないことを保証したりする厳密な検証である．

前者のアプローチの有効性は十分にあると考えているが，鉄道の安全を考える立場としては，特に証明による仕様の整合性の保証という点に強い関心を持っている．システムが定められた機能を有していることを数学的に証明することで，システムの安全性を高められると考えている．そこで，定理証明技術の信号保安装置の仕様等に適用する検討を本論文の主題とする．

ここで対象とする信号保安装置は，前述の基本的な装置のうち，電子化にはなじまない転てつ装置（機械部分が多く，電力も必要）や信号機（条件に従って灯を点灯する機能があればよく，扱う電流も大きい．信号機へ出力する制御までを考慮しておけば十分である）を除く，閉そく装置，連動装置，ATS（ただし本文では詳細には取り上げない），ATC といった範囲の装置とする．同種の装置に踏切保安装置があり，高い安全性が要求されるものの，踏切保安装置に適用されている技術は，踏切に障害物が存在することを検知する踏切障害物検知装置を除くと，連動装置や閉そく装置と同様の部分が多い．そのため踏切保安装置そのものは本論文では取り上げないが，本論文の応用として適用可能である．

これらの装置の処理の大部分は，数十〜数百 msec 毎に確定した入力を用いて論理演算を行うものとなる．部分的にはそれに加えて数値計算を行うが，ATS 信号（数十 kHz〜数 MHz）や ATC 信号（数 kHz）のアナログ波形を直接変復調する部分を除くと，周期毎の入出力の変化が定義される誤差よりも小さいとみなせる範囲での計算となる．実際には入力が急激に変化する場合があるが，その変

化部分については過渡現象として扱われ，誤差そのものは重要とされない．

これらの装置やそのサブシステムを検証するには，対象とする装置を形式仕様記述言語で記述し，検証できるモデルを作成しなければならない．この点に関し，本論文ではモデル化の過程を示し，併せてその際に適用した工夫等を示すことで，信号保安装置の検証モデルが作成可能であることを示す．さらに実際の検証過程を示すことで，信号保安装置が検証可能であることを示す．

また，実際には定理証明の産業システムへの応用事例そのものが限られており，適用事例から得られる知見そのものが少ない．例えば B メソッドと呼ばれる手法が成功したことが宣伝されているが，実際に適用を試みると大量の証明責務が生成されてしまうなど様々な困難に直面する．そこで単に定理証明の適用を目的とするのではなく，そこから得られる知見に関して考察を加えることで，信号保安装置以外の分野であっても有効な，定理証明等の活用に関する知見を得ることについても本論文における課題とする．

本論文の構成は以下のとおりである．まず第 2 章において信号保安装置に適用される安全性技術について概説する．続いて第 3 章においてフォーマルメソッドに関して，モデル規範型手法を中心に，定理証明技法やモデル検査技法について応用面に焦点をあててサーベイを行う．この現状を踏まえ，第 4 章において，対象となるモデルや使用する手法の選択について述べる．

第 5 章では定理証明による検証の前に，フォーマルメソッドによるモデル化の習熟を兼ね，フォーマルメソッドのもう 1 つのアプローチである Light Weight なアプローチについて，VDM（Vienna Development Method）[1] を用いた連動装置のモデル化を行い，その意義を議論する．

第 6 章から第 8 章では実際に定理証明を適用した 3 つの事例について述べる．この 3 つの事例は信号保安装置の広い範囲をカバーするように選択している．

第 6 章では，静的なデータの検証として，ATC システムにおける線区データベース仕様の証明による検証を実施した．ここでは研究段階の VDM 仕様の証明機能を使用した．ここでは仕様記述だけでなく証明機能の GUI（Graphical User Interface）上においても VDM の表記を用いるが，バックエンドに the HOL system[2] を使用している．実際にはこの証明機能の開発は研究段階で止まってしまい，実用化がなされなかったが，証明に関する知見については現在でも有効であると考えている．

第 7 章では，動的なシステムのうち歴史的にリレーで論理が組み立てられてきた装置の検証として，単線区間の運転方向の制御（閉そく制御）をモデル化し，証明を行った．手法としては B メソッド [3] を使用し，最終的にコードの生成までを実施した．この事例では，条件分岐の使用により証明責務が大量に生成されるため，これを軽減するための対策について検討を行った．

第 8 章では，リレーによる論理では対応できない数値計算の事例として，ATC システムの許容速度（ブレーキ曲線）の計算に B メソッドを適用した．この事例では仕様の実装段階において証明責務が大量に発生している．この発生の理由を探ることで数値計算への B メソッドの適用可能性を議論する．

そして第 9 章において，これらの知見を総括し，今後の展望について述べる．

第 2 章

信号保安装置における安全性技術

まずはじめに信号保安装置における安全性技術について概説する．

2.1　信号保安装置の解説

冒頭で述べたように，信号保安装置とは鉄道の安全な運行を支える装置群であり，以下の機能を担うものである．

1. 列車同士を正面衝突させない
2. 後続の列車を先行する列車に追突させない
3. 駅構内においてポイント（線路を 2 方向に振り分ける装置）において列車を側面衝突させない
4. 鎖錠（ロック）されたポイントを割出し（開通していない方向から車両が進入する）たりしない

これらの機能に加えて，分岐部や曲線などで速度が上がりすぎて脱線するのを防ぐために，速度を制限する機能などがある．具体的にこれらの機能を担うのが列車検知装置，閉そく装置，信号機，転てつ装置（ポイントを動かしたり，鎖錠したりする装置），連動装置，ATS（Automatic Train Stop : 自動列車停止装置），ATC（Automatic Train Control : 自動列車制御装置）といった装置である．そのほかに道路と線路が交差する箇所に設備される踏切の制御のための踏切保安装置があるが，ここでは説明を省略する．

まず，列車を追突させないために，初期の鉄道で採用されたのが時間間隔法（隔時法）である．これは先行列車に対して後続列車を一定時間おいた後に出発させるという方式で，現在でも路面電車で使用されている．しかしこの方式では，先行列車が長時間停止した場合に追突する可能性が高くなる．そこで考案されたのが空間間隔法と呼ばれるものであり，先行列車と後続列車に対して一定の空間間隔をあけて運転する（図 2.1）．空間間隔法の実現のために，駅間を閉そく（block）と呼ばれる区間に分割し，各閉そく内には 1 列車のみに占有させるという方法を採った．なお，図 2.1 に ATS 地上子の記述があるが，これは運転間隔を制御するためのものであり，詳細については後述する．

この閉そくにおいて列車の在線を検知するものが軌道回路（Track Circuit）と呼ばれる列車検知装置である（図 2.2）．これはレールに電流を流し，列車の輪軸の有無の違いで電流の流れ方が変わることで列車を検知するものである．このアイデアは 1872 年に特許として成立しているが，常時電流を

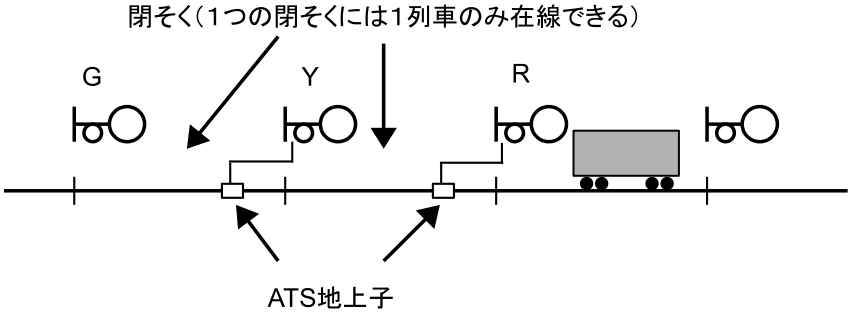
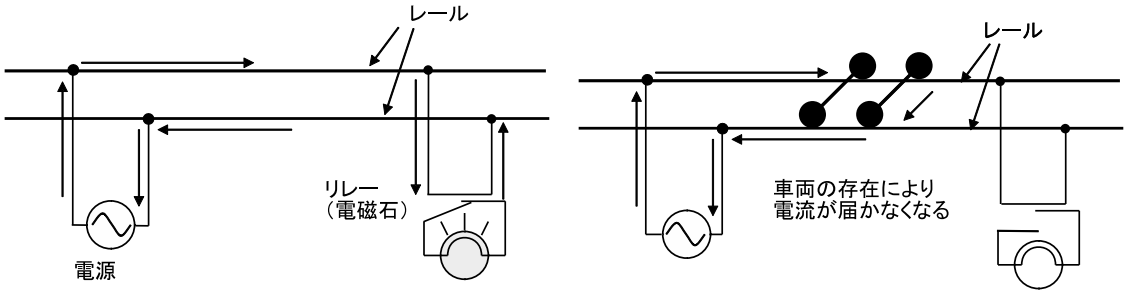


図 2.1　駅間における信号保安装置（閉そく）



（a）非在線時　　（b）在線時

図 2.2　軌道回路（閉電路）

流してコイルを励磁させておき（図 2.2 （a)），列車が在線することでコイルに電流が流れなくなると，励磁がなくなるために接点が開放される（図 2.2（b)）閉電路と呼ばれる仕組みが使われている．励磁がない場合に接点が開放されるのを保証するために，重力やばねを仕組みに取り入れている．

軌道回路では一部の例外（踏切を開ける制御のための検知など）を除き，閉電路の仕組みが原則である．それは電源が故障したりレールが折れたりした場合でも，無励磁になり列車が在線している状態と等価になることで，軌道回路が故障して在線が実際には検出できないような場合であっても列車を進入させない仕組みが作れるからである．軌道回路の仕組みとして，通常は回路が開放状態となり，輪軸の存在により回路が構成されると電流が流れる開電路と呼ばれる方式も存在するが，この方式では故障時に列車を検知できなくなるため，列車を検知できないことが安全側となる一部の例外（踏切を開ける制御など）に用途が限られている．

また信号の現示[*1]には古くは腕木式信号機が使われた．この信号機は，信号機を操作するてこと接続されたワイヤにより信号現示を行うが，ワイヤが切れたり，リンク機構が壊れたりすると停止現示になるように作られた．

これらのものを見ても信号保安装置には古くからフェールセーフ（Fail Safe）の概念が確立していたことが分かる．フェールセーフとは装置の一部に故障があっても危険側に制御されないような仕組みである．その後，信号保安装置には様々な改良や新しい装置が導入されるが，そこには常にフェールセーフの考え方が貫かれている．これを実現するために重要な性質が非対称誤り特性，つまり故障したときの機器の状態が 2 つの状態のうちの特定の一方となる性質である．例えば，リレーは電源が

[*1] 信号機が灯などで示す符号，つまり運転士への指示を現示と呼ぶ．

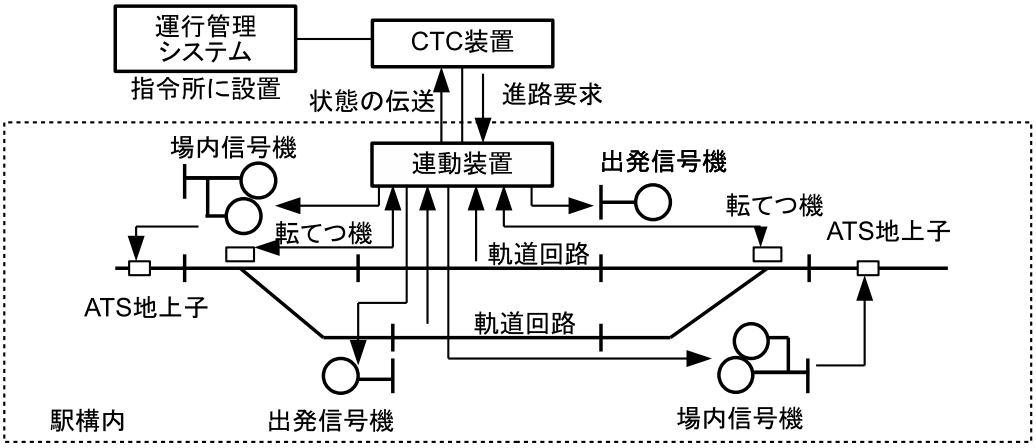


図 2.3　駅構内の信号設備の概略

供給されなくなるなどの故障の場合は，励磁状態となるよりは無励磁状態となる可能性が高い．そこで無励磁状態が安全側になるように機器を設計することで安全が確保される．

駅構内の信号保安装置の構成を，図 2.3 に示す．列車の走行経路である進路というものを定め，それぞれに信号機を対応させた．競合する（経路が重複する）進路が同時に設定されないように，進路に対応した信号機やポイントを動かすてこ（操作のためのレバー）に鎖錠関係（互いにロックされること）を設け，進路が設定されると競合する他の進路を設定できないようにした．これが連動装置と呼ばれるものであるが，最初はこれを機械的なリンクで実現していた．やがて論理に用いるリレー（継電器）が実用的になり，連動装置はリレーで構成されるようになった．これが継電連動装置であり，現在でも広く使用されている．その後，次節で解説する電子化技術（ME（Microelectronics）化技術）により，電子連動装置が登場し，機器室のスペースの節約や高機能化に貢献している．

連動装置は元々は各駅に設置され，信号扱者が操作を行っていたが，現在では CTC（Centralized Traffic Control：中央列車制御）装置により遠隔操作が可能となり，指令所から集中的に制御を行うことが可能となった．さらに CTC 装置の上位に PRC（Programmed Route Control：自動進路制御）など，駅の進路自動制御機能が設けられ，その結果，運行管理業務は進路制御のための操作から解放され，運転整理等に専念できるようになった．CTC や PRC は併せて運行管理システムと称されているが，基本的には円滑かつ効率よい運行のために存在するものであり，現在でも基本的には各駅に連動装置が設置され，上位の運行管理システムに対して列車の衝突や追突を防ぐための最後の砦となっている[*2]．

列車の間隔の制御には図 2.1 に示したように軌道回路に連動させた信号機を用いている．列車検知状態に従い，列車が在線する閉そくの入り口では停止信号を現示するが，その 1 つ手前の閉そくでは軌道回路電流を逆向きに流し，受信器において軌道回路電流が逆向きであることを別の接点により検出することで，注意信号（すなわち黄信号）を現示する．

さらに安全性を高めるために実用化されたのが ATS である．もっとも古い ATS に打子式 ATS というものがある．この方式では，停止現示の場合，停止現示に連動して地上の打子（trip arm：ハンマーのようなもの）が立ち上がり，これにより列車の下部に設けられたブレーキコックが開けられ

[*2] 集中連動装置という概念もあり，連動装置の論理処理部分が各駅にではなく，中央などに集中化されている場合もある．

る．すると車両のブレーキ管に込められた圧縮空気が開放され，それによりブレーキが作用する．これは機械的に実現されたものであるが，同様のことを電気的に行えるものが後に開発された．日本では，車上に 100kHz 前後の発振回路を構成し，その発振信号のフィルタ出力を増幅・整流してリレーを励磁させる．この発振回路が，停止現示に連動した地上装置（地上子と呼ぶ．図 2.1 や図 2.3 のように配置される）と電気的に結合すると，発振周波数が地上子の発振周波数に変化し（変周すると呼んでいる），その結果車上のフィルタ出力が低下し，それによりリレーを落下させる．これが変周式という呼ばれる仕組みであり，現在も使用されている．発振回路とフィルタという組み合わせを使うことにより，発振回路が故障してもブレーキがかかる仕組みとなる．これにより運転士が停止信号に対して確認扱いを行わないと自動的にブレーキがかかるシステムが構築された．これが国鉄で採用された ATS-S と呼ばれる仕組みであり，1962 年の三河島事故を契機に全国に導入されている．

このシステムの最大の欠点は運転士が一旦確認扱いを行うとシステムの防護が働かなくなる点であるが，これに対して駅の入り口等で停止現示の時に必ず止める必要がある場合（絶対信号と呼ぶ）には，即時停止の信号を設けて確認扱いなしでブレーキをかける機能などが追加されている（ATS-Sn と呼ばれる）．

その後変周式に代わり，ディジタル電文を地上に設置された伝送装置（地上子）と車上の伝送装置（車上子）との間でやり取りするトランスポンダ（transponder）を利用することで，停止距離を直接車上に与える ATS-P と呼ばれるシステムが 1987 年に実用化され，保安度の向上だけでなく，列車間隔の縮小にも役立っており，特に首都圏などで広く導入され，現在に至っている．

ATS は運転士がブレーキをかけ忘れたときのバックアップ装置から発展してきたが，積極的にシステムで車両の速度を制御するのが ATC である．ATC では，レールに列車に許容される速度に対応する信号を流し，それを車上の装置が解読して，許容速度より現在の速度が超過していればブレーキをかけ，許容速度以下になるまで制御する．信号機は一部の例外を除いて設置されず，その代わりに運転台に許容速度が表示される車内信号式が使用されている．また，このシステムでは軌道回路を使って信号を伝送しているが，そこでは 1kHz 前後の搬送波を速度信号に対応した変調周波数で変調した信号が使われた．変調回路や増幅回路などに半導体が使われるが，沿線に送受信器を設置するには環境条件が劣悪であることもあり，ATS においては駅間に分散して設置された軌道回路の送受信器についても，ATC においては信号機器室に集中的に配置されるようになった．このシステムにおいて信号がない無信号状態であれば許容速度が 0km/h と解釈され，非常停止する制御が行われている．これにより ATC 地上装置において故障が発生したり，列車検知状態に異常が発生すると，装置からの信号の送信を止めることで列車を停止する機能を持たせることが可能となっている．新幹線においては ATC 装置は架線の電源と連動しており，架線の加圧がない場合には信号が停止する仕組みとなっている．この仕組みは，地震発生時に変電所を停止することで列車を停止させる早期地震検知システムにも活用されている[*3]．

なお，このような機器を製作するのは現在でこそ困難ではないが，ATC が開発された 1960 年代においては，まだ半導体でさえ実用化初期の段階でふんだんに使えるものではなく，信頼性についても決して高いと言えなかった．発振回路も LCR といった受動素子を中心にしており，安定性も悪い．そのこともあり，ATC 装置は最初から多重系としていた．また送受信器の設置場所について信号機

[*3] 変電所が停止すると，ATC が停止し列車を停止させる．ただし実際には ATC とは関係なく車両が停電を検知するとブレーキがかかる仕組みとなっている．

器室に集中させる機器集中方式を採用したことはすでに述べたが，当初個別素子のトランジスタで実装されていたものが集積回路となり，性能が安定してきた現在でも，ATC 装置においては当初の思想を引き継ぎ，多重系の構成，機器集中方式（例外もあるが）を採っている．

現在の信号保安装置は，軌道回路による列車検知結果を基本に制御を行うのが基本となっている．しかし，これを車上による位置検知機能と無線による通信を組み合わせて制御するシステムが検討され，導入が始まっている．このようなシステムでは列車位置の確定，間隔制御などはソフトウェアなくしては実現できないものとなっている．

2.2　電子化技術における安全性技術

元々はリレーで構築されてきた信号保安装置であるが，マイクロプロセッサが登場してくると，これらを用いて信号保安装置を構築できないか模索が始まった．ここで問題となるのが，マイクロプロセッサを構成する論理素子が非対称誤り特性を持たないことから，果たして装置をフェールセーフにできるのかという点である．

非対称誤り特性を持つフェールセーフ素子が発明され，これを論理素子として用いて，論理演算回路を構築する手法も研究されたが [4]，そのためには，全ての論理演算をフェールセーフ素子とする必要があり，これが特殊な実装となることから，汎用計算機の進歩に追い付かず実用的な論理回路とはならなかった．実用的に電子化機器をフェールセーフとするためには，対称誤り素子を前提としたフェールセーフ構成手法が必要になる．この課題については，理論的には 1980 年代後半に完成している．

入力，出力が一定の符号語となる回路について考える．回路に故障が発生した時に非符号語の出力を採ることはあっても誤った符号語を採らない回路をフォールトセキュア（FS : Fault Secure）と呼ぶ．また故障が発生した時，通常の入力により非符号語が出力される回路をセルフテスティング（Self Testing）という．回路がフォルトセキュアかつセルフテスティングである回路をトータリーセルフチェッキング（TSC : Totally Self Checking）回路と呼ぶ [5]．この回路は単一故障に対する性質であるが，複数の故障に対して回路の出力の正当性を保証するのがストロングリーフォルトセキュア（SFS: Strongly Fault Secure）という性質である [6]．一方，符号語入力に対して必ず符号語を出力し，非符号語出力に対しては必ず非符号語を出力する回路をコードディスジョイント（CD: Code Disjoint）と呼ぶ．

これらの概念は元々が組合わせ回路に対して定義されたものであるが，これを順序回路を含む一般の論理回路に拡張し，「誤り安全」「誤り伝搬性」[7] の概念を導入することで実用的に SFS 性を確保する方法が定式化された．システムをいくつかのサブモジュールに分割し，それぞれが SFS 性を満たすように構成し，サブモジュールが CD でない場合にはその入力インターフェースにおいて TSC 回路を置いて符号語を検出すれば，全体として SFS 性を得られる [8] という結論が導かれた．そしてこの考えに基づいて汎用プロセッサを用いた構成法が提案されている．その後も各種回路が提案されているが，それでもなお汎用のプロセッサと比べると回路が複雑なのが問題となる．

これに対して，国鉄では実用的なシステムとして，冗長構成を用いたバス同期システムというものを開発し，1985 年から連動装置に適用して使用開始している [9]．図 2.4 に実際に適用された装置の構成を示す [10]．

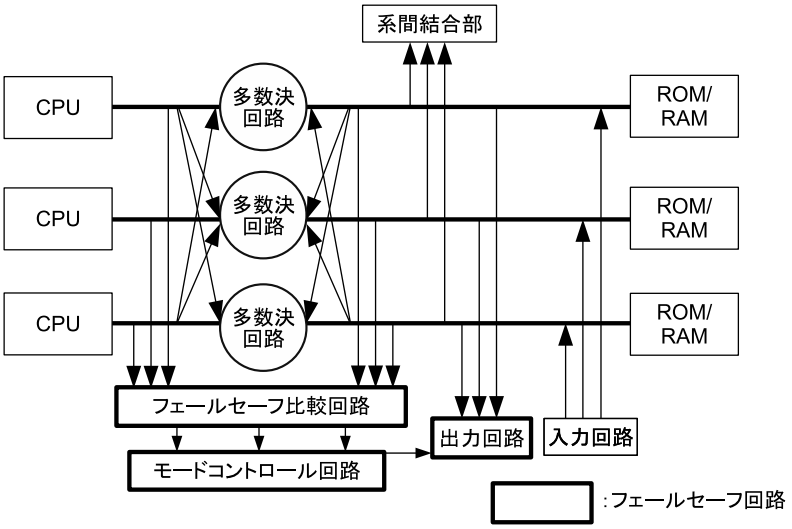


図 2.4　バス同期式フェールセーフ計算機の構成

3 つのプロセッサは共通クロックにより同一の処理を行う．バスデータについては 2 つずつのプロセッサのバスラインをマシンサイクル単位で比較を行い，不一致検出時には機器への制御出力を遮断する．比較回路にはフェールセーフに構成した振り子回路と呼ばれる回路を採用した．これは 2 つのフリップフロップを用い，不一致が発生すると出力が 0 固定となる回路である．このような多重系と比較回路の組み合わせにより SFS 性を確保した．このシステムは演算素子については汎用 CPU（Central Processing Unit：中央演算処理装置）を用いることができることから比較的容易に実装が可能である．出力部についてもフェールセーフ素子を用いた多数決回路を構成して，現場機器に対して危険側情報を出力しないようにしている．また，入力回路については，3 重系に構成して，それぞれ各系の多数決バスに入力されるように構成した．さらに危険側の故障を検出するために定周期で診断を行っている．

このように，信号保安装置においては多重系を構成してバスレベルで比較する方法が採用され，現在に至っている．入力については，多重系で入力する手法のほか，入力データを出力し，再入力して故障を診断する手法も使用される，出力についても，フィードバック入力して正しい出力がなされているかをチェックする手法などが採用されている．

連動装置への電子化機器の導入と並行して，電子化技術が導入されたのが ATC 車上装置である．ATC 車上装置についても，1964 年の東海道新幹線開業当初は個別素子の半導体とリレーの組み合わせで構成され，フィルタなどの処理も受動素子を中心としていた．その後，演算増幅器などのアナログ IC（Integrated Circuit）が利用できるようになり，改良はされたものの，機能増大とともに大型化の一途をたどっていた．それが 1985 年の東海道新幹線 100 系向けの ATC 装置においてはじめて本格的に電子化されることで小型化に貢献し [11]，現在につながっている．これらの電子化機器を鉄道分野では ME（Microelectronics）化機器と呼んでいる．

ATC 地上装置についても 1990 年前後から電子化された装置が導入されはじめ，現在に至っている．従来の装置は速度信号毎に信号波の発振器があり，それをリレーで選択して軌道回路毎に増幅を行っていた．また，速度信号の決定も在線状況からリレー論理で決定していた．機器集中方式だからこそ採用できる方式であったが，現在では，信号は個々の送信器において DSP（Digital Signal

Processor）で波形を生成している．速度信号の判定もリレーによる選別ではなく，ソフトウェアによって判断する状況となっており，論理部についても小型化される結果となっている．最近ではATC 装置と連動装置を一体化した装置 [12] も導入されつつある．

2.3 ソフトウェアに関する安全性技術

実際には，前節で示したように信号保安装置には既にソフトウェアが多く入っている．では，これらのソフトウェアの信頼化をどのように確保しようとしているのかを見てみる．

連動装置が最初に電子化された当初はアセンブラが使用されていた [13]．装置の安全性を確保するために，ソフトウェアに関しては安全性に関して重要な vital な部分と，そうでない non-vital な部分を分けることで，vital な部分におけるバグの混入を減らすことが重要だとされた．また，モジュール化を行った上で，各モジュールの大きさを極力小さくする方針が採られた．また，各駅に適用可能な連動論理を処理する標準プログラムと，駅毎の個別設計に相当する駅データを分離する構成としている．これにより，標準プログラムで論理処理の安全性や正当性が確保されれば，駅毎の連動装置の設計については駅毎のデータの設計に専念すればよいこととなる．

この場合は，個別設計部分がどうなるかが問題となる．最初に開発された電子連動装置の駅毎のデータについては，シュプール方式と呼ばれる方式が採用されている [13]．これは，連動図（図 2.5 参照．これと連鎖を記述した連動表を併せて連動図表と称する）に基づいた線路の接続関係を表現したシュプール図と呼ばれるモデル（図 2.6 参照）を作成し，そのノード部分に連動の条件を記述するものである．連動図においては，縦線で区切られた範囲が軌道回路に対応し，それぞれに AT，BT，…などの名称がつけられている．線が分岐している箇所が転てつ器（列車の進行方向を振り分ける装置）であり，そのそばに示された 51 イ，51 ロ，52 などの文字列はその名称である．また信号機のシンボルが進路に対応している．これがシュプール図においては四角で囲んだ部分がユニットと呼ばれる単位であり，ここに隣接ユニットとの接続関係やそのユニットに関する条件などが記述されている．これを列車が出発する箇所（発点と称する）から到着する箇所（着点と称する）までをたどって予約することで進路が確保される仕組みとなっている．当時はメモリ容量や CPU の処理能力に限りがあったことから，規模が大きくなってもコンパクトな表現ができるこのような手法が採用されたが，それまでの継電連動装置の設計との乖離が大きい上，標準プログラムで対応しきれない機能も多かった．

現在ではメモリ容量の制約が小さくなったことから連動図表を直接マトリクス表現，つまり進路同士の連鎖や進路と転てつ器の間の連鎖を 2 次元のテーブル（実際には複数のテーブルを用いる）で表現し，それに従って処理することで，制御を実現するマトリックス方式と呼ばれる手法や，連動図表をリレーの結線図と対応可能な論理式に変換し，その論理式を処理する結線入力方式 [14] と呼ばれる手法が広く採用されている．また，JR 東日本では連動表自体を見直し，発点から着点までの軌道回路を順番に予約することで連動論理を実現する軌道回路予約方式 [15] も使用されている．いずれも処理プログラムと個別データが分離されていることに変わりはない．

信号保安装置の処理を行うコードの記述については，現在では C 言語が多く使用されている．その作成の仕方について見てみる．信号保安装置向けのソフトウェア技術に関して明文化された資料としては文献 [16] がある．これはコンピュータの機能安全に関する国際規格 IEC61508（IEC：

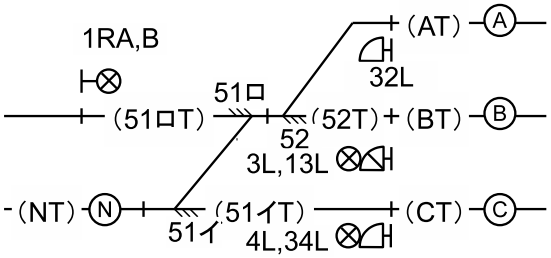


図 2.5　連動図の例

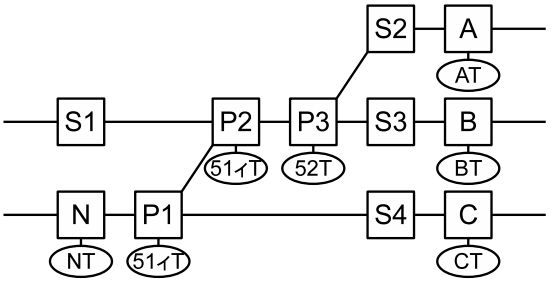


図 2.6　シュプール図のイメージ

International Electrotechnical Commission 国際電気標準会議）に合わせて作成された文書であり，鉄道用ソフトウェアに関する国際規格 IEC62279 に取り上げられている個別技術もほぼ同様となっている．

ここでの基本的な考え方は次のとおりである．

1. 安全性が要求される機能とそのほかの機能を分割し，安全性が要求される部分を単純化する．
   （a）モジュール化設計，プログラムの構造化を行う．
   （b）go to 文を禁止する．
   （c）シングルスレッド処理とし，定周期タイマで起動する．
   （d）安全側と危険側でプログラムにおける情報を明確に分ける．これは具体的には論理値 0 を安全側に，論理値 1 を危険側に割り当てることを意味している．
2. 異常試験，安全性試験，現用設備との動作比較のためのモニタラン試験を行う．

つまり，プログラムにコーディング規約を設けた上で，処理をシンプルに行う方法が基本であり，電子連動装置が開発された当初からの考え方を踏襲している．

一方，異常検出方式には様々な手法が用いられている．

1. ウォッチドグタイマを用い，暴走状態になった場合に異常を検出する
2. CPU の相互割り込み起動（複数の CPU 間で相互に周期的に割り込みをかける）
3. プログラムモジュールの起動順序チェック
4. アクセプタンステスト（入力に関して，チェックを行った上で受け入れる）
5. ダブルリンク（ファイル相互の関係を順方向，逆方向の両方で正当性を検証できるように構成する）
6. 画面へのフリッカ出力（画面の更新が止まることで異常を検知できるようにする）
7. メモリプロテクト（決められた範囲を超えてメモリアクセスをしようとするのを検出する）
8. データに通番等の情報をつける

これらを見ると分かる通り，プログラムがバグの有無，論理素子故障のいずれの原因かを問わず暴走してしまったことに対する検出方法については様々な対策が行われている．また誤りが混入しにくいように考慮もされている．一方で，ほとんどの手法がプログラムそのものが正当であることを示す手法にはなっていないことが分かる．スウェーデンで実用化された最初の電子連動装置においては N バージョンプログラミング（ソフトウェアダイバーシティ）が採用されたが，このような技法が広く

採用されているとは言えない．

プログラムの正当性をいかに確保するかは非常に重要な問題である．現状ではテストやレビューによっていると言っても過言ではない．連動装置に関しては連動検査法が JIS E 3004（JIS：Japan Industrial Standards 日本工業規格）で定められており，連動装置に搭載する連動図表データのチェックについて方法が確立している．実際に連動装置設計支援機能としてこのチェックリストに基づいた検査の自動化もなされている．ただし，この検査方法は過去の経験に基づいたテストベースの手法である．従ってチェックした内容に関しては問題がないことが示されるが，チェックリストの内容が十分であることの正当性は厳密には証明されておらず，より信頼性の高い手法が望まれる．また，連動装置とは異なる新しい装置については検査手法を新たに作る必要が生じる．場合によっては経験則によれない可能性も高い．その場合にチェックリストに正当性を与えるのは連動装置以上に困難であると考えられる．

これに対してプログラムの開発手法で誤りを減らす方法が考えられるが，ここで注目したいのがフォーマルメソッドである．フォーマルメソッドとは，システム仕様のなんらかの形式化を通じてシステムの信頼性の向上を図るものである．例えばその中には定理証明や自動検査による検証手法があるが，この適用によりさらに高いレベルの正当性が示せれば品質向上が図れるものと考えられる．

定理証明やモデル検査といったフォーマルメソッドの技術を導入することにより，信号保安装置のプログラムの品質向上を図ることができないか，その可能性をこれから検討していく．

第 3 章

フォーマルメソッドによるソフトウェア開発の現状

前章の最後にフォーマルメソッドという言葉が出てきた．

フォーマルメソッドと呼ばれる手法にも多くがあり，VDM や B といったモデル記述に重点を置くモデル規範型，CSP（Communicating Sequential Processes）などのプロセス代数，あるいはシステムのモデルの性質を網羅的に検査するモデル検査法などがある．万能的な手法はないが，特に我々はプログラムの正当性を与える手法に着目している．すなわちシステムを記述した後，それが正しいかどうかがチェック可能で，コード生成もできるという点からモデル規範型手法が有効であると考えている．

本章ではフォーマルメソッドによるソフトウェア開発手法について，後半部分において使用する VDM や B メソッドなどのモデル規範型手法を中心に解説する．また，本論文に対する関連研究についても手法の解説に合わせて各節で紹介する．

3.1 VDM による開発

VDM（Vienna Development Method）は，IBM ウィーン研究所におけるプログラミング言語 PL/I コンパイラの正しさを形式検証する研究に端を発する，最も古いフォーマルメソッドの 1 つである．PL/I 言語定義のメタ言語として Meta-IV が開発され，それが現在の VDM につながっている．

過去には VDM で記述した仕様の証明の検証も行われており [17]，証明支援系の mural なども存在したが，現在では CASE（Computer Aided Software Engineering）ツールとしての側面が強くなっている．

VDM という言葉には，VDM-SL などの仕様記述言語と，それらの言語を使用してシステムを開発する手法の両方の意味がある．VDM の仕様記述言語には，ISO 標準（ISO : International Organization for Standardization 国際標準化機構）である VDM-SL（Specification Language），それを大規模開発に適用可能とすべくオブジェクト指向とした VDM++，さらにリアルタイム処理，組み込み系への適用を目指した VDM-RT（VICE）の 3 種類がある．VDM-SL と VDM++ では，構造化の仕組みが大きく異なるが，基本的な記法については大きく変わることはない．ここでは

```
types
  T1 : nat
  inv t == t < 50;
  T2 = <A> | <B> | <C>;
  T3 = token;

state vars of
  v1 : T1
  v2 : T2
  v3 : T3
inv mk_vars(v1, v2, v3) == f(v1, v2, v3)
init v == v = mk_vars(0, <A>, mk_token("C"))
end

functions
  f: T1 * T2 * T3 -> bool
  f(p1, p2, ..., pn) == ...
  pre ...

operations
  Op(a1: T1, a2: T2, ..., an: Tn) ret:R
  ext wr v1: T1
      rd v2: T2
  pre  p(a1, ..., an, v1, v2, v3)
  post q(a1, ..., an, ret, v1~, v2~, v3~, v1, v2, v3)

values
  x : T2 = <B>
```

図 3.1　VDM-SL のモジュールの概要

VDM-SL の記述を中心に解説する．

3.1.1　VDM の文法の解説

VDM-SL のモジュールの構成を図 3.1 に示す．ここでは単一モジュールを前提とした記述であり，構造化については後述する．types で型の宣言を行い，state では状態変数（state variable）をまとめて宣言する．演算は 2 種類ある．状態変数に引数を用いて変更を加えるものを operation （操作）と呼ぶ．一方，引数のみから，何らかの値を返すものを function（関数）として区別している．引数は値の参照であり，演算の中で値を変更はできない．それぞれ operations，functions に続けて記述する．types，operations，functions は 1 箇所にまとめる必要はない．そのほかの構成要素として定数があり，values の箇所で宣言する．定数は function においても参照可能である．

型

VDM における全ての変数は何らかの型（type）を有している．VDM の基本型は 整数型（int），自然数型（nat），ブール型（bool），文字型（char）のほか，シンボルをかぎ括弧（< >）でくくって識別できるようにした引用型（quote）や，詳細を決めず値の区別のみが可能なものとして定義されるトークン型（token）がある．引用型においては | （合併型，union）を組み合わせることで複数の引用型の値を取る仕組みとなる．トークン型の値は ISO 標準では一致不一致の比較のみができるが，VDMTools の拡張で識別子を与えることができる．

これらの基本型に対し，集合（set）や列（seqence），複数のフィールド値を組み合わせて構成するレコード（record）型，2 つの集合の一方（domain）の値にもう一方（range）の値を対応づける写像型（mapping）などを使うことで複雑な値を表現できる．

集合は`{ }`，列は `[ ]` の中に要素をコンマで並べ，`{1, 2, 3}` のように記述する．また，内包的記法（comprehension）を使用できる．例えば

```
{a | a : nat & a < 10 and a mod 2 = 0} = {0, 2, 4, 6, 8}
```

型 `A` の集合は `set of A`，列は `seq of A` という型を持つ．

レコード型の宣言では，`::`のあとに構成子（フィールド）を宣言する．例えば 2 次元の位置を示す座標型 `location` を例にすると以下のように記述できる．

```
location :: x : nat
            y : nat
```

このような記述を行うと，`location` 型の変数 `loc` の要素は `loc.x` のように参照可能となる．`mk_location(1,2)` などという形で `location` 型の値を構成することも可能となっている．

型 `A` から型 `B` への写像型は `map A to B` という型を持つ．個々の対応関係はマップレット（maplet）`{a |-> b}` を用い，`{1 |-> true, 2 |-> false}` のように記述する．

型に対して，その型に属する値が常に満たす必要がある条件を不変条件（invariant）として定義することができる．例えば -30 度以上 100 度未満の値を示す温度型 `Temp` は次のように定義できる．

```
Temp = int
inv t == t >= -30 and t < 100
```

inv が不変条件を示すためのキーワードであり，その次の `t` は不変条件の記述に使う仮変数である．不変条件を満たすように演算を記述するというのが VDM の記述に要求される事項である．レコード型では各フィールド間の条件を不変条件に記述することが可能で，これにより複雑な制約を定義できる．

集合や写像はフォーマルメソッドにはよく見られる概念であるが，フォーマルメソッドに触れていない人にとっては，プログラムを書くのに慣れた人ほど，理解しにくいもののひとつである．形式仕様記述言語の集合には順序という概念がないが，プログラムの記述ではループ変数を使った配列の演算を記述する場合が多いためか，集合の演算の理解が難しいようである．

演算子

VDM では bool 型（`true`，`false` を値として取る）に対する演算子として，論理積（ASCII 文字では `and`），論理和（`or`），否定（`not`），含意（`=>`）等が提供されている．数値に対しては不等号（`<`, `>`, `>=`, `<=`）および等号（`=`）が比較演算子として提供される．これらの演算子は 2 つの入力（否定は 1 入力）に対し，bool 型を返す function として扱われるため，論理式や比較演算子で構成される述語は bool 型を持つことになる．数値に関してはそのほかに加減乗除の演算子などがある．VDM は LPF（logic of partial functions）の立場を取っており，言語仕様上は真理値が決定しない（undefined） 命題を扱える．例えば `a = true`，`b = undefined` とした場合，`a and b = undefined`, `a or b = true` となる．ただし，ツールでは実用上の問題から C 言語と同じように，先頭から評価を行い，評価値が定まった時点で評価を打ち切る処理を行っている．

集合に関しては，所属（∈，ASCII 文字では `in set`），部分集合（⊂，もしくは `subset`）が bool 型を返す演算子として提供されており，そのほかに集合の和（∪，もしくは `union`），積（∩，もしくは `inter`）といった演算子が提供されている．さらに，一階述語論理の全称限定子 ∀（ASCII では `forall`），存在限定子 ∃（同じく `exists`）があり，bool 型を返す述語を構成できる．

状態変数

VDM-SL ではシステムを記述する状態変数については一箇所で宣言を行う．記述は以下のようになり，状態変数 `v1`，`...`，`vn` の型および変数間の不変条件（`inv` 以下），初期化に関する記述（`init` 以下）を行う．

```
state vars  of
  v1 : t1 ...
  ...
  vn : tn
inv  s == ...
init i == ...
end
```

VDM++ ではモジュールの単位となるクラスの中にクラス変数（instance variables）を記述する．クラスごとにクラス変数間の不変条件を記述することになる．

演算の記述

operation および function についてはそれぞれ，演算の具体的な中身を記述する explicit（陽）な記法と，演算前と演算後の状態変数あるいは引数の関係のみを記述する implicit（陰）な記法がある．

explicit function は以下のように記述する．

```
f: T1 * T2 * ... * Tn -> T
f(p1, p2, ..., pn) == expression
pre P(p1, p2, ..., pn)
```

ここで p1, p2, ..., pn は引数であり，T1, T2, ..., Tn はその型である．expression は引数を使って具体的に返り値を表現するものである．例えば 2 つの値のうちの大きい方を求める関数 max は次のように記述できる．

```
max: nat * nat -> nat
max(a, b) == if a > b then a else b
```

P は事前条件と呼ばれるものであり，p1, p2, ..., pn の述語（真か偽かを返す関数）である．引数の値は P が真になる範囲に制約される．なお，explicit function において，まず a に代入して，続いて b に代入するといった逐次演算を直接記述することはできないが，let 文（後述）を使うと，式に対する置換表現ができ，if then else 文もあるので，逐次演算に近い表現は可能である．

一方，implicit function は具体的な演算の中身を示さずに引数と返り値と間の性質だけを記述するものであり，返り値を ret （別の名前を用いてもよい）とすると，以下のように記述される．

```
f(p1:T1, p2:T2, ..., pn:Tn) ret:T
pre  P(p1, ..., pn)
post Q(p1, ..., pn, ret)
```

P は explicit function と同様の事前条件であり，真の場合に初めて function が適用可能となる．Q は事後条件と呼ばれるもので，計算前の p1, p2, ..., pn の値，および返り値 ret との間で成立する条件を記述する．

operation では，状態変数も含めて記述を行う．implicit operation は次のように記述する．

```
Op(a1:T1, a2:T2, ..., an:Tn) ret:R
ext wr s1: ST1
    rd sm: STm
pre  P(a1, ..., an, s1, s2)
post Q(a1, ..., an, ret, s1~, ..., sm~, s1, ..., sm)
```

ext に続いて宣言されている s1, ..., sm は状態変数であり，これらに対して参照（rd）および値の変更（wr）を行うことを宣言する．事後条件の記述では，引数，返り値および operation 実行前の各変数および実行後の各変数の値の関係を示す．ここで operation 実行前の各変数については~を付けて参照する．operation に関しては事前条件，事後条件とも引数だけでなく，状態変数を含めた記述が可能となっている．

explicit operation では，以下のように記述する．

```
Op : T1 * T2 * ... * Tn => R
Op(p1, p2, ..., pn) == statement
```

ここで statement は代入等を示すものであり，基本的には逐次演算の記述を行う．代入（:=）のほか，operation 呼び出し，if 文や case 文，for ループ（整数をループ変数とするほか，集合や列の要素に対し処理を行うこともできる），while ループが提供される．また，並列実行や非決定的実行（演算の順序が非決定的となる）などについても，制約はあるが使用可能である．基本的には逐次演算を

記述する場合は，逐次実行する各文をブロック文を意味する括弧で囲む．

VDM の本来の考え方では，仕様においては何（What）を演算するのかを記述すべきで，具体的なアルゴリズム（How）を記述するのではないとされ，implicit な表現の方が VDM らしいとされる．しかし explicit に記述した方が簡潔にできる場合もあり，後述するツールでの実行には explicit な記述が必要であるため，必要に応じて使い分けることになる．

let 文

VDM では表現を簡潔にするための，ローカルな定義による置換表現が可能である．それが `let` 文である．

```
let a = f(x1, x2, ..., xn) in g(a)
```

とするとローカルに `a` が定義でき，`f(x1, x2, ..., xn)` の代わりに `a` を使用してそれ以降の文を記述できる．なお，`let` 文の中に状態変数を用いる場合には同様の意味を持つ `def` 文を用いる．

さらには非決定的に値を選択する `let be st` 文がある．構文は以下のとおりである．

```
let a in set X be st f(a) in g(a)
```

こうすると集合 `X` のうち `f(a)` を満たす値を非決定的に選択し，その値を用いてそれ以降の演算を記述できる．

構造化

ISO 標準の VDM-SL では，基本的に 1 つのモジュールで全ての仕様記述を行うこととし，大規模化は考慮されていない．これに対し開発環境である VDMTools では独自の拡張により，構造化への対応をしている．module と呼ばれる概念を導入し，module 内に，型や function，operation，状態変数を定義する．さらに他の module へ export する型や function，operation を宣言することで，これらを他の module に import 可能としている．ただし，状態変数についてはそれぞれの module 内に閉じられ，外部の module から参照することはできない．なお，module 名を宣言しない場合には `DefaultMod` が暗黙に宣言されるようになっている．

VDM++ では基本的なモジュールの構成単位はクラスとなり，クラスの内部にインスタンス変数，function（引数のみを用いて，返り値を得る演算）および operation（引数のほかにインスタンス変数を参照，変更できる演算）を定義する．VDM++ では型の不変条件のほか，インスタンス変数間の制約として記述される不変条件を記述できる．

VDM-SL の記述はシングルスレッドとなっているが，VDM++ ではオブジェクト毎にスレッドを生成可能となっている．

3.1.2　VDM の開発環境について

現在 VDM の開発環境は 2 つある．1 つは VDMTools[18] でかつてデンマーク IFAD 社が開発・提供していた VDM-Toolbox と呼ばれるツールを日本の CSK 社（現在 SCSK 社）が買い取り提供しているものである．もう 1 つは Overture Tool[19] であり，オープンソースプロジェクトのツール

である．こちらは Java の開発等に使用される eclipse のプラグインとして実装されている．なお，実際には両者は対立関係にあるわけではなく，技術面での協力が行われている．

VDMTools では，VDM 仕様記述の文法チェック，タイプチェックのほか，インタプリタによる仕様の実行やデバッグが可能となっている．実行の際は不変条件や事前事後条件を動的にチェックすることが可能で，実行時に不変条件が満たされない場合に警告を発するようにできるが，これらの条件のチェックの計算時間が短いことが必要で，条件（特に forall などで複数の変数を束縛変数として使用した場合など）によっては制御不能に陥る．証明による検証機能は提供されていないが，仕様の正当性を保証するために証明が必要な項目（証明責務: proof obligation）を生成する機能は提供されている．

仕様の実行においては，基本的に explicit な表現のみが実行できるが，implicit 部分については，外部のプログラムにその実装部分を委ね，その結果に対して事後条件をチェックするような使い方で評価可能となっている．また，インタプリタの利用はツール上で対話的に行うだけでなく，スクリプトにより実行することも可能で，テストスクリプトを作成することにより，系統的なテスト実行を可能としている．コードカバレージ（記述を実際に評価したか否か）の計測機能があり，インタプリタで実行して評価した部分に色付けすることが可能である．

テストという観点でなく，仕様を実行して，その結果をわかりやすい形でユーザに示すアニメーションが重要とされており，そのための手段として CORBA （Common Object Request Broker Architecture）API（Application Interface）[20] を使った外部プログラムからのインタプリタとの通信機能が提供されている．この機能については第 5 章で使用しているので，その中で実際の使い方を示す．

VDM++ の記述においては UML（Unified Modeling Language）との連携が可能になっており，UML 記述から，VDM++ の記述を得たり，逆に VDM++ 記述の修正を UML 記述に反映させたりする形で使用可能である．

なお，テストに関しては最低限の支援機能が提供されているという状況であり，どのようにテストするかはユーザにゆだねられている．ただし，Test Suite を提供しようという動きはあり，VDMTools のサポートページで紹介されている．

3.1.3 VDM による開発事例

当初は VDM はシステムを証明により検証することを狙っており，VDM を使った形式検証の研究が行われていた．VDM の言語仕様が定まると共に証明に関する理論などの研究もなされた．文献 [21] はその理論面を解説したものである．また文献 [17] には証明での検証事例が取り上げられている．その中には核処理施設の追跡システムや弾薬倉庫の管理システムのモデルも含まれている．これらの例は手作業による証明を行っているが，別の章では証明支援系 mural[22]，PVS[23]，Isabelle[24] を用いたものについても言及されている．mural とは 1990 年前後に開発されていた証明支援系であり，Smalltalk により実装されていた．PVS, Isabelle（現在はその上に高階論理（Higher Order Logic）を構築した Isabelle/HOL が使用されることが多い）はいずれも現在も広く使用されている証明支援環境である．

一方，1990 年代から 2000 年初め頃，IFAD 社において VDM-Toolbox が提供されていた頃には，

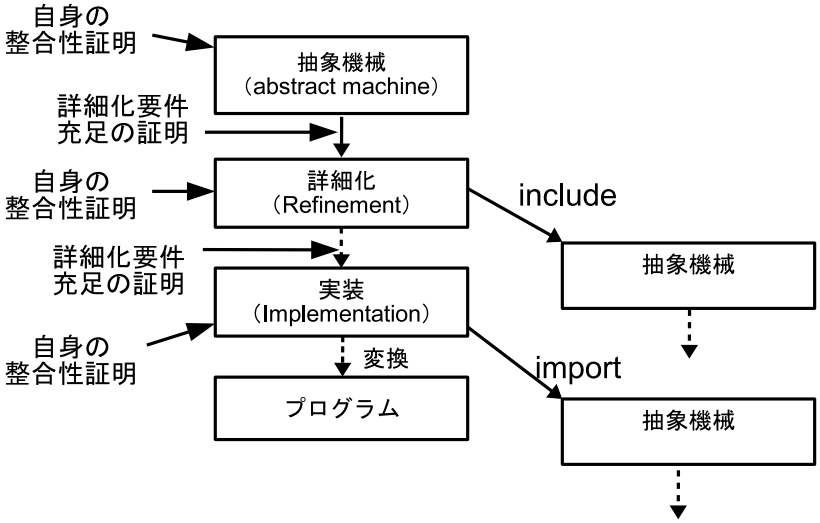


図 3.2　B メソッドによるプログラム開発

IFAD 社が VDM を活用したシステム開発のコンサルティングを行っていたこともあり，多くの産業分野への適用事例が生まれている．例えば，空港管制センターの表示端末 [25]，廃棄物処理システム [26]，花卉市場のオークションシステム [27]，紙幣の処理装置 [28] などが紹介されている．ただし，VDM-Toolbox は証明の機能を持っておらず，こういった適用事例では証明を用いていない．

日本国内では証券会社のバックオフィス向けパッケージソフトの開発 [29] や IC チップの設計の例 [30] が挙げられている．前者ではそのサブシステムである「マル優管理システム」の詳細設計に VDM++ を使用している．本来要求仕様として記述すべき制約式について検討を行い，それにより品質向上に役立ったとしている．後者では，IC チップのファームウェアにおけるデータ構造を定義するファイルシステム仕様，検証のためのフレームワーク，プロトコル仕様やセキュリティ仕様について記述している．また同時にテスト仕様やテストスクリプトを別の担当者によって記述し，テストを実施している．コード生成器は使用しておらず，仕様の策定を中心に VDM を使用している．これらの事例についても定理証明を行っているわけではない．

VDM の最近の適用事例は定理証明を行うというよりも VDM のラピッドプロトタイピング機能を活用した開発が中心と言える．厳密な証明を用いた検証についてはその機能が提供されなかったこともあり，最近は例が見られない状況にある．

3.2　B メソッドによる開発

B メソッドとは J. R. Abrial によって提唱された手法であり [3]，フランスを中心に使われているモデル規範型手法である．検証済みコードの生成に重きを置いており，仕様から生成される定理の証明と，仕様を段階的にコードにまで詳細化するという概念をもったものである．

3.2.1　B メソッドの文法の解説

B メソッドでのシステム開発イメージを図 3.2 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

```
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 3.3 B メソッドの抽象機械モジュールの概要

記法 [31] 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特徴がある．抽象という言葉が示す通り，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可能である．また変数および変数間の関係について常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 等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，証明器等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 3.2 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していることを証明する必要がある．この部分は図 3.2 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 3.3 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 3.3 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 3.3 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」と言っているのは，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，ブール型あるいはその配列のみが許される．

また定数も使用できる．宣言は CONSTANTS に続く箇所にて行う．定数の型については変数と同様な型を持つことができるが，定数であるため途中で値を変更することはできない．定数の型は PROPERTIES の中で記述する．実装しない抽象定数も可能であり，その場合は ABSTRACT_CONSTANTS に続いて宣言する．

INVARIANT は不変条件を示す．変数の型の宣言のほか，変数の間に常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 実行後には常に満たされている必要がある．

型

B の基本型としては，整数型やブール型（BOOL: TRUE および FALSE），文字列型（String）がある．整数型は無限の整数の集合（Natural，Integer）と処理系で表現できる範囲の有限の整数の集合（NAT，INT）とに区別される．実装段階では Natural や Integer は使用できない．VDM の引用型に相当するものは B では enumerated set と呼ばれる．SETS の部分で set 名とともにその要素を宣言する．変数の型については，変数の宣言部分で示すのではなく，INVARIANT に続く不変条件の中で宣言する．型の解釈のない変数は認められていないので，INVARIANT 内で必ず宣言する．これにより，operation 実行後の値が変数の型を満たしているかどうかが証明責務として生成される仕組みとなっている．

VDM のレコード型に相当する構造体も宣言はできるが，構造体のフィールド変数同士の関係についても INVARIANT の中で記述することになる．構造体に名前がつけられないため扱いにくく，変数が組として取り扱えるという以外にメリットがないため，B では構造体は多用されない．

ある型 A の値からなる集合は，A の部分集合であるため，型としては A のべき集合（Power Set）型となる．べき集合に相当するキーワードは POW であり，型 A の値の集合 B は不変条件内で `B : POW(A)` のように記述できる．

VDM の写像型に相当するものは，B では一般化されている．集合 A の値 a と集合 B の値 b の組 (a, b) の集合を A と B の relation と呼んでいる．そのうち，a の値に対し，組になる b の値が一意となるものを function と呼ぶ． a の集合が A と一致すれば全関数，A の部分集合であれば部分関数と呼ばれる．さらに単射や双射も定義可能である．このあたりの考えは Z の記述を引き継いでいる．

列については列挙した場合の記述に VDM と同じように [] を用いることができる．ただし列に関しては内包表現は使用できない．A の要素からなる列は `seq(A)` という型を持つが，実際には列は順番を示す整数から値への関数として定義されており，`seq(A) : POW(Z * A)` となる．

述語を扱うこともできるが，述語の型の定義はなく，ブール型とは明確に区別されている．述語をブール型として扱うためには Bool 演算子を用いて変換する．例えば `Bool(A = B) = TRUE`．VDM と異なり，TRUE, FALSE を論理演算子の項には用いることはできない．

実装段階においては具象変数のみが使用可能である．この具象変数に関してはプログラムに変換可能な範囲に変数の型が制限されており，具体的には変数は整数型（deferred set を含む），ブール型，

enumerated set しか使用できない．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．

演算子

演算子に関しては，VDM と比較すると，ASCII 表現に若干の違いがあったり，演算子そのものが多く用意されていたりする（特に集合に関しては多い）が，個々の演算子を見ると大きな違いはない．

述語に関しては，論理和（∨，ASCII 文字では `OR`）論理積（∧ あるいは `&`），否定（¬ あるいは `not`），含意（⇒ あるいは `=>`），等号（＝あるいは `=`），不等号（≠ あるいは `/=`，≤ あるいは `<=`など），所属（∈ あるいは `:`），部分集合（⊆ あるいは `<:`，⊂ あるいは `<<:`など）全称限定子（∀ あるいは `!`），存在限定子（∃ あるいは `#`）が提供されている．ただし，これらはあくまでも述語を記述する際に用いるものであり，前述したようにブール型の値に対して論理和，論理積，否定，含意は使用できない．

集合に関しては和（∪ あるいは `\/`），積（∩ あるいは `/\`），差（-）などの演算子があり，列に関しては連結（`^`を使用）や最初の要素（first），最後の要素（last）などの演算子がある．

一般化代入

初期化や operation の中で演算として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 3.1 に主なものを示す．それ以外に条件選択など，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 3.3 の `aa :: AA` では `aa` に `AA` の 1 要素を非決定的に代入する意味になる．`||`は複数の代入を同時に行うことを意味するが，同時に同じ変数への代入はできないので，その場合は条件選択などを併用する必要がある．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．

なお，VDM における function のような，引数を用いて値を返す演算は用意されていない．同様のことを直接行いたい場合は，マクロかラムダ式を用いるしかない．ただしマクロでは引数の型のチェックができないため，完全に VDM の function と同じ記述はできない．

特に VDM では bool 型を返す function を事前事後条件から呼び出すことが許され，その結果を論理演算子で他の function や論理式と結合できるので，事前事後条件の一部を別の function とすることが容易にできるが，B では operation 呼び出しはあくまでも一般化代入の一つであり，事前条件代入や非決定的選択の条件内で使用することはできず，さらに述語とブール型が明確に分けられているために，条件を分けて記述するのには困難が生じる．

VDM の function に相当するものを実装するには，operation と対になる変数を定義することで対応する方法があるが，この方法については第 8 章で使用しているので，その場面で説明する．ただし，この方法でも VDM の事前事後条件と全く同じ記述を行うのは困難である．

構造化

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 3.3 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になる．include するモジュール `Minc` の前に prefix をつけずに宣言すると，operation や変数の名前が自分自身の変数や

表 3.1　B における一般化代入の例

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
有限の非決定的選択	`CHOICE S1 OR ... Sn END`	`S1,...,Sn` の 1 つを非決定的に選択
条件選択	`SELECT P1 THEN S1 WHEN P2 THEN S2 ... ELSE T END`	成立する `Pi` に対応する `Si` を非決定的に選択
IF 条件	`IF P1 THEN X1 ELSIF P2 THEN X2 ... ELSE Y END`	通常の IF〜THEN と同様
非決定的選択	`ANY X WHERE P(X) THEN S END`	`P(X)` を満たす `X` を用いて `S` を実行
逐次代入	`A; B`	`A` を実行した後，`B` を実行
同時代入	`A || B`	独立した代入を同時に行う
operation 呼び出し	`xx <- Op(aa, bb, ...)`	他の operation を実行
事前条件代入	`PRE P THEN Q END`	`P` が成立するときのみ `Q` を実行
ブロック代入	`BEGIN ... END`	逐次代入や同時代入をブロック内で使用

operation と衝突した場合にエラーとなるが，例えば図 3.3 に示したように `SS` と prefix を付けると，この operation や変数を図 3.3 の `SS.Op_C` のように `SS` を付けて参照することが可能となり，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．なお，定数はモジュール間で共通なので，prefix をつけずに参照する．

インスタンスの値の変更は外部モジュールで定義された operation を通じて実施する必要がある．また operation での振る舞いはあくまでも include する抽象機械の定義によるため，そ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．

そのほかに sees リンクや uses リンクがある．sees リンクは他のモジュールで import されたモジュールを参照するというものである．この場合，モジュールの変数を参照できるのみなので，operation を通じた値の変更は許されていない．uses リンクは同一モジュールに include される複数モジュール間でデータを共有するための仕組みである．

上に挙げたリンクのいずれに対しても，モジュール間の各項目にアクセスできるかどうかが定義されている．この制約は証明を行うために注意深く構築されており，厳密に守る必要がある．

証明責務

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

$$I \Rightarrow [S]I \tag{3.1}$$

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，例えば 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

$$[C]\neg[B]\neg J \tag{3.2}$$

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`，詳細化後が `PRE Q THEN L END` の場合に

$$(I \wedge J \wedge P) \Rightarrow (Q \wedge [L]\neg[K]\neg J) \tag{3.3}$$

で与えられる．また include リンクを行う場合には，前提に include する抽象機械の不変条件が加わる．

3.2.2 B メソッドの開発環境について

かつては B メソッドの開発ツールとしてイギリスの B-Core 社が B-Toolkit を提供していたが，現在ではサポートを行っていない．一方，フランスの Clearsy 社が Atelier B[32] を提供している．かつては学術向けの B4free というツールを並行して提供していたが，現在では Atelier B 自体をフリーとして一本化している．なお，フリーで提供されるバージョンのほかに保守契約を結ぶことによって入手可能な保守契約バージョン（実際には最新版となる）がある．

Atelier B においても GUI（Graphical User Interface）環境が提供されており，文法チェックおよび証明責務の生成，自動証明および対話的証明が行えるようになっている．また詳細化支援ツールとして BART（B Automatic Refinement Tool）が提供されているが，一方で VDM にあるようなインタプリタは提供されていない．

しかしながら B のアニメーションツールとして，デュッセルドルフ大学で提供している ProB[33] がある．このツールでは B で記述された仕様について，任意の実行可能な operation を選択して実行することが可能で，各ステップごとに変数の状態を確認することができる．また不変条件やデッドロックが発生するかどうかを網羅的に状態遷移を発生させて検証することも可能となっている．

ProB は直接 Atelier B とリンクはしていないが，BEval という拡張機能が開発されており，これを通じてリンク可能となっている．（ただし，Linux 版のみ提供）

Atelier B は 32bit 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 32bit 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後述する．

3.2.3 B メソッドによる開発事例

B メソッドの適用事例はフランスを中心に多くみられる．特にパリ地下鉄 14 号線（Météor）[34] や空港シャトル（VAL）への適用 [35] が有名であり，単に鉄道分野への適用ということに留まらず B メソッドの大規模システムへの適用例として紹介されている．文献 [34] については，システム開発報告の色彩が強く，プロジェクトの進め方や定量的な評価を行っていることが特徴となっている一方，具体的なモデル化手法や証明の技術にはあまり触れていない．一方，文献 [35] においてはモジュールの構成の考え方やモデルの記法について言及されていて，B を使う上での示唆を含んでいる．

そのほかにはホームドアの制御 [36] やスマートカードへの適用事例 [37] などが報告されている．

なお，B はどちらかというとシステムのモデリングや開発に適用するよりは，より低レベルの機能のプログラム作成に向いており，研究としては次節に紹介する Event-B の方がさかんとなっている．

```
MACHINE <machine identifier>
REFINES <machine identifier>
SEES <context identifier>
VARIABLES <variable identifier>, ...
INVARIANTS <label>: <predicate> ...
THEOREMS <label>: <predicate> ...
EVENTS <event>
VARIANT <variant>
END
```

図 3.4　Event-B の MACHINE 定義

```
CONTEXT <context identifier>
EXTENDS <context identifier>, ...
SETS <set identifier>, ...
CONSTANTS <constant identifier>, ...
AXIOMS <label>: <predicate> ...
THEOREMS <label>: <predicate> ...
END
```

図 3.5　Event-B の CONTEXT 定義

3.2.4　Event-B の文法の解説

B メソッドの派生となる Event-B[38] について解説する．Event-B は，B メソッドにおける検証済みコードの生成に対し，システムのモデル化に重点を置いた手法である．

Event-B のモジュールはマシン（MACHINE）とコンテキスト（CONTEXT）の 2 つに分けられる．図 3.4 および図 3.5 にモジュールの構成を示す．コンテキストには集合（carrier sets），定数（constants）およびそれらが満たす公理（axioms），定理（theorems）が記述される．いわば静的な部分の定義である．定数の定義は定数名を `CONSTANTS` で宣言し，型については `AXIOMS` で定義する．また carrier sets は set の名前を `SETS` で宣言する．その性質については `AXIOMS` で宣言するが，B における enumerated set の宣言にそのままは対応していない．そこで登場するのが partition というブール値関数で `partition(S, s1, ..., sn)` と記述すると，これは `S` という set が集合 `s1, ..., sn` で構成されるという意味になる．この場合に `s1, ..., sn` に重複がないことが要求される．これを用い，`s1, ..., sn` で列挙する値を宣言すれば，enumerated set が構成できる．

`AXIOMS` では公理を示す述語，`THEOREMS` では定理を示す述語を記述する．各述語にラベルを付ける所が B と異なるが，記法としては B の述語と同様である．

これに対してマシンではコンテキストを参照(see)し，変数(variables)やその不変条件(invariants)，変数を書き換えるイベント（event）が記述される．また，コンテキストの定理・公理やと不変条件から導かれる定理（theorem）を記述できる．こちらは動的な部分の定義となる．変数名は `VARIABLES` で宣言し，型は `INVARIANTS` で宣言する．

状態遷移に相当するイベントは，図 3.6 の形をとる．`ANY ... WHERE` というキーワードを使っていて B メソッドの非決定的選択に似ているが，predicate と action に label が付いている．`WHERE` に続く文がいわばガードという形になっていて，これがイベントが発生する条件となる．パラメータがない場合には `ANY ... WHERE` ではなく，`WHEN` を使う．`THEN` に続くのが action で，この中では変数の代入等が行われる．しかし，Event-B では使用できる代入が整理されており，B の単純代入，要素代入，充足代入しか認められていない．そのため，B の CHOICE や SELECT などを使った記述はできないが，充足代入ではイベントの事前事後の値を使った記述が可能である．`STATUS` はイベントの収束性に関する宣言を行うためのオプションである．

```
<event identifier>
STATUS {ordinary, convergent, anticipated}
REFINES <event identifier>, ...
ANY <parameter identifier>
WHERE <label>: <predicate> ...
WITH <label>: <witness> ...
THEN <label>: <action> ...
END
```

図 3.6 イベントの定義

マシンは詳細化（refine）することができる．一方，コンテキストは拡張（extend）される．どのマシンを詳細化したか，どのコンテキストを拡張したかはそれぞれ REFINES や EXTENDS に示す．1 つのマシンを複数のマシンに詳細化できるが，逆に複数のマシンを同時に 1 つのマシンに詳細化することははできない．

B メソッドと異なり，implementation に相当する部分がないので，最終的なコード生成を目指す形とはならないが，詳細化が行われているかのチェックについては証明可能であるため，システムのモデル化や検証を目的にするのであれば十分実用になるといえる．

3.2.5 Event-B の開発環境および開発事例

Event-B での開発については Atelier B でのサポートもなされているが，主に eclipse 上で動く RODIN プラットフォーム [39] での提供となっている．RODIN プラットフォームでは Event-B 記述の支援や証明の支援環境が提供されている．plug-in によって機能拡張が行える仕組みとなっている．

証明支援は Atelier B Provers plug-in によっているが，それとは異なる証明支援系についても plug-in として提供が行われている．Event-B では公理系を拡張するための Theory Plug-in という仕組みがあり，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 [40]，実数に関する標準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 [41]．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．

RODIN そのものはコード生成を扱っていないが，Event-B にコード生成機能を追加しようとする研究 [42] も行われている．

Event-B はどちらかというとコード生成よりはシステム記述に重きを置いているため，実用例といってもシステムレベルの検討が中心となる．ケーススタディが文献 [38] 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 [43, 44, 45] などが報告されている．

この中で文献 [43] 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．

3.3　証明支援系による検証

VDM 仕様の検証で出てきた PVS や Isabelle/HOL などの証明支援系そのものについては，仕様をそれぞれの記述言語で記述すれば証明による検証が可能になる．しかしながら多くの証明支援系では高階論理を用い，関数型プログラミングによる仕様記述が求められ，帰納的な定義も多用される状況にある．VDM や B のようなモデル規範型と比べ，より数学的な記述が求められるため，要求されるレベルが高くなる．より高度な成果が期待できる一方，一般的なソフトウェア開発プロセスに導入するのは困難で，このような手法を導入するには形式検証に特化した専門家集団が必要とされるものと考えられる．

また，多くの場合仕様を記述するだけでは不十分である．それは記述された仕様に対する証明責務を作成する必要があるためである．また，VDM-SL などの言語で記述された言語を証明支援系を用いて検証する場合は，仕様そのものだけでなく，仕様記述言語に関する定理や公理を準備し，さらに証明責務を生成して支援系に投入することが必要である．これは支援系単独で行う場合でも同様である．例えば Isabelle/HOL の場合，他の開発言語における仕様記述を，証明のために Isabelle/HOL に変換する事例は Z 記法を対象とした HOL-Z[46] や Event-B における Isabelle Plugin など主要な形式記述言語に対しては見られるものの，Isabelle/HOL での関数は全関数を前提としており，基本的に停止関数のみが記述できることから，Isabelle/HOL による記述に対する証明責務生成器という議論は見られない．

また，PVS や Isabelle/HOL ではコード生成があまり考えられていなかったが，最近になって，Isabelle/HOL から Haskell や Scala といった言語への変換器が登場している．ただし，鉄道保安装置のプログラムでは C 言語が使用されることが多いため，関数型言語に出力される場合は，さらに C 言語への変換が必要である．

3.3.1　Isabelle/HOL

Isabelle/HOL はケンブリッジ大の Paulson とミュンヘン工科大の Nipkow によって開発された証明支援系である．汎用の証明支援系 Isabelle に高階論理 HOL（Higher Order Logic）を形式化して構築された．なお，the HOL system[2] と呼ばれる別の高階論理に基づく証明支援系（これは第 6 章の VDM 証明機能のバックエンドとなっている）の影響を受けている．Isabelle は論理を形式化するためのメタ論理であり，直観主義高階論理に基づいた型付きラムダ計算である．Standard ML（Meta-Language）[47] を用いて構築されている．

その上に構築する HOL はチャーチによる型理論（simple theory of types）に基づいたものであり，Isabelle と同様に型付きラムダ計算に基づいている．型としては bool や nat といった基本型や $\tau \Rightarrow \tau$ と記述される関数型がある（ただし nat は実際には帰納的に定義されたものである）．これから作られる項には，引数を適用した関数のほか，ラムダ式 $\lambda x.t$，さらには条件式 `if b then t1 else t2` などがある．そして bool 型を持つ命題が定義できる．`True` や `False` といった定数のほか，通常の論理演算子（$\neg, \wedge, \vee$のほか, 含意 $\longrightarrow$, 等号 $=$，限量子 $\forall, \exists$）などが定義されている．

帰納的な関数やデータ型の定義もできる，それによりリストや自然数の型が定義されるが，停止関

数である必要がある．また関数は全関数である必要がある．

Isabelle では，支援環境として Emacs 上で行う古典的な Proof General と，統合開発環境（IDE : Integrated Development Environment）である Isabelle/jEdit が提供されている．前者の主要な機能は証明スクリプトを管理することにある．後者は GUI で提供されている．

Isabelle の証明は実装言語である ML のレベルで推論規則の適用（tactic）を行う方法と，Isar と呼ばれる，人が記述する証明に近い証明記述言語を用いる方法がある．前者を使う場合は tactic が ML の関数として実装されているのを用いる．tactic を適用する毎に証明の実行状態を更新していく．定理や公理を追加したり，推論規則に従って新しい命題を導き，最終的な命題にまで持っていく Forward Proof と，最終的な命題を複数の副命題（subgoal）に分割していき，全ての subgoal が真となるまで，tactic を適用していく Backward Proof のやり方がある．一方，Isar を用いる場合は Backward Proof となるが，より人間が記述する証明に近い形となる．**lemma, theorem** といったコマンドにより証明モードに入る．**proof** rule，**apply** rule という形で定理を適用したり，**show** で補題の証明を行ったりする．自動的に証明する **by** auto のようなコマンドもある．このようなコマンドは統合環境で対話的に行うが，最終的に証明できるとそのスクリプトと共に保存される．この際に自由変数はスキーマ変数と呼ばれる変数に変換される．これは代入可能な変数であり，これにより再利用可能となる．

Isabelle/HOL の適用事例も数多い．数学の問題の検証もあるが，プログラム（仕様）の検証事例としては Java Virtual Machine のモデルの検証 [48] やより大きな規模な事例としてプロセッサのマイクロカーネルへの適用 [49] などがある．

3.3.2 Coq

Coq はフランス国立情報学自動制御研究所（INRIA）で開発されている証明支援系である．1984 年から開発されている．同じ INRIA で開発されている OCaml（ML の別の実装）で実装されている．

Coq が基礎とする論理は Calculus of Inductive Constructions と呼ばれる直観主義論理であり排中律などが成立せず，古典論理を基礎に置く Isabelle/HOL と対比される．その記述に使用される記述言語は Gallina と呼ばれる型付きラムダ式である．Gallina では，整数やブール値が集合（Set）の型を持つ datatype として定義されている．例えば

```
Inductive bool : Set :=
  | true : bool
  | false : bool

Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.
```

命題については Prop 型という型を持っている．また, Prop 型や Set 型が属する型として Type 型が定義されている．Prop 型，Set 型，Type 型をソート（Sort）と呼んでいる．

関数はキーワード `fun` を用いてラムダ式として定義できるが，これに名前を付けることができる．また入力に依存して出力の型が決定する依存型関数も記述でき，これにより表現力が増しているとされる．

Coq も Isabelle/HOL と同様，対話的に実行を行う．Gallina では Vernacular というコマンド群を有している．datatype や関数の定義，定理の宣言なども Vernacular の一部である．証明は Proof というコマンドによって開始される．その中では tactic と呼ばれる推論規則の適用を繰り返し，証明すべき命題（Goal）を副命題（Subgoal）に分割して，最終的に命題を証明する．証明ができると tactic の実行列であるスクリプトとともに，名前がつけられて保存される．

Coq では，Curry-Howard 同型を利用している．これは命題が型，証明がプログラムと対応しているということを意味していて，証明を手作業で行うことと，目的の型を持つ式を構築することが同等となる．よって，命題としてプログラムの仕様，すなわち性質が記述された場合に，それが証明されると，その証明に対応したコードが生成できる．直観主義論理に基づくことから排中律や背理法が使えず，証明が困難となったり，場合によっては不可能となったりする場合があるが，その代わりに証明からコードを生成することが保証できる．

出力言語は OCaml が標準であるが，そのほかに Haskell や Scheme にも出力可能である．いずれも直接出力できるのは関数型言語であるが，これらの言語処理系には C 言語へのコンパイラが存在するので，C 言語のソースに変換可能となる．

Coq は四色問題の形式検証を行ったことで話題となった [50]．数学における定理の証明への使用も活発である．一方，プログラム検証では，C コンパイラ CompCert の形式検証 [51] の例が有名である．プログラムの形式検証の場合，これらは C のソースを検証するということではなく，Coq（Gallina）で形式記述を行い，必要な定理を証明して検証した後，プログラムを出力する形をとる．従って，もとの仕様をいかに記述できるかが鍵となる．Compcert での形式検証は現在も続いているが，このことは結局何を検証すべきかということが問題となることも意味している．

3.3.3　PVS

PVS（Prototype Verification System）は SRI Inernational 社が開発した証明支援系である．こちらも高階論理に基づくものであるため，関数を関数の引数として取れるなどの点は Isabelle/HOL や Coq と同様である．

ただし，名前に Prototype が冠されているところからも見て取れるが，仕様記述という点に注目して言語仕様を定めている点で，これまでの 2 つのシステムと大きく異なる．1 つは解釈のない型（Uninterpreted Type）を導入しているという点であり，VDM のトークン型のように，実体を決定しない形で型を定義し，型に関する制約だけを記述することを可能としている．もう 1 つは Sub Type を導入しているという点であり，つまりある型をもとに，その部分集合を型として定義できるが，ここで Sub Type の定義に述語を用いることが可能となっており，VDM における型の invariant に近い記述が可能となっている．さらに関数として部分関数を認めている点が Isabelle/HOL や Coq と異なっている．部分関数を認めることの副作用として，関数適用が実行できることに関する証明責務が発生するが，これはシステムの方で TCC（Type Check Condition）として生成する．

仕様の記述例を図 3.7 に示す．これは証明器の解説書 [52] から引用したものである．**THEORY**

```
sum: THEORY
  BEGIN
    n: VAR nat
    f, g: VAR [nat -> nat]
    sum(f, n): RECURSIVE nat =
  (IF n = 0
     THEN 0
     ELSE f(n-1) + sum(f, n - 1)
   ENDIF)
   MEASURE n

  sum_plus: LEMMA
     sum((lambda n: f(n) + g(n)), n)
        = sum(f, n) + sum(g, n)
  square(n): nat = n*n

  sum_of_squares: LEMMA
    6 * sum(square, n+1) = n * (n + 1) * (2*n + 1)
   cube(n): nat = n * n * n
   sum_of_cubes: LEMMA
   4 * sum(cube, n+1) = n*n*(n+1)*(n+1)
END sum
```

図 3.7 PVS の記述例

がモジュールの宣言に相当する．その中で型，変数，関数，公理（**AXIOM**），命題（**LEMMA, THEROEM**）を宣言する．関数においては VDM にあるような IF～THEN～ELSE や LET 文が使えるため，逐次実行を直接記述することはできないまでも，仕様記述としてはよりモデル規範型に近い記述ができるものと思われる．

仕様記述後，型チェックを実行してパスすると TCC や命題を証明することになる．証明はやはり Backward Proof となる．前提（antecedents）および結論（consequents）を構成する各命題を sequent と呼ぶ．個々の sequent には番号が振られるため，証明のコマンドはその番号をパラメータとして与える．tactic に似たコマンドが多く用意されているので，それを対話的に入力していく．論理式の複雑さを緩める **flatten**（$A \wedge B$ が前提にあれば，代わりに A, B を前提の sequent とする，$A \vee B$ が結論にあれば，代わりに A, B を結論の sequent とする，$A \Rightarrow B$ が結論にあれば A を前提の sequent とし，B を結論の sequent とするなど）や **split**（$A \wedge B$ が結論にあれば，代わりに A, B を結論の sequent とするなど），**skolem**（スコーレム化，つまり限定子の変項を自由変数に置き換える）などのコマンドがあるほか，他の命題などを利用することも出来る．

PVS の適用事例のサーベイとしては少し古くなるが文献 [53] があり，ハードウェア検証，アルゴリズム検証の例が報告されている．2000 年代に入ると，PVS を使用しているという報告は決して多くはないものの，現在でも NASA（National Aeronautics and Space Administration:アメリカ航空宇宙局）が継続的に利用しており，実数計算を含むライブラリを充実させ，小型航空機輸送システムの例 [54] などに適用している．

3.4　モデル検査法

証明による検証とは別のアプローチとしてモデル検査法（model checking）が注目を浴びている．これはシステムを状態遷移モデルの形で記述し，それに対して到達性や変数の条件などの調べたい性質を変数の値を網羅的に変化させて調べる手法である．システムを形式的に分析するという点において，モデル検査法もフォーマルメソッドによる検証手法の 1 つとして扱われている．

SPIN[55] や SMV[56], LTSA，UPPAAL など様々な手法があるが，システムを記述する変数が増えると状態数が爆発して調べきれなくなる．あくまでも状態遷移モデルに対する検査手法であり，モデルが自動でプログラムに変換されるわけではないため，プログラムに誤りのないことを証明できるものではないが，適用が容易で結果が明確に出力されるという利点もあり，適用例は多く鉄道分野でも文献 [57] などの報告がある．

ここでは代表的なモデル検査法である SMV と SPIN についてその概説と適用事例を示す．

3.4.1　SMV

モデル検査法の提唱者は E.Clarke とされる．彼らは時相論理の 1 つである CTL （Computation Tree Logic：計算木論理）を用い，システムの Kripke モデル $M = (S, R, L)$ について，システムが満たすべき性質を CTL 式 f とした場合に

$$s \in S \mid M, s \models f$$

（論理式 f を満たす状態の集合）が初期状態を含むかどうかを判定することをモデル検査の問題として定式化した．

その後カーネギーメロン大学にて SMV（Symbolic Model Cheker）[56] と呼ばれるツールが開発される．このツールでは記号モデル検査が使用できる．これは，状態を 1 つ 1 つ列挙するのではなく，論理式を満たす値の集合により状態を表現することとした．個々の状態を明示せず，集合として記号で表現するためにそう呼ばれる．その上で BDD（Binary Decision Diagram: 二分決定木）と呼ばれるデータ構造を採用し，これにより論理演算の効率化が図られた．

現在は SMV の後継として NuSMV[58, 59] が使用されている．これは SMV を再実装し，拡張したものである．NuSMV では BDD によるモデル検査のほかに有界モデル検査（BMC : Bounded Model Checking）が提案されている．記号モデル検査でも扱えない対象に対して，全てを探索しなくても不具合が発見できることが多いことが経験的に分かってきたことから，初期状態の集合に対して，指定したステップ数までの範囲で状態の探索を行うのが BMC である．技術的には BDD ではなく，与えられたステップ数までの状態集合および検証する命題を論理式で表現し，これを充足可能性判定器（SAT ソルバ）を使って判定するというものである．

SAT ソルバの SAT とは Satsifiability Problem を指しており，2 値変数からなる論理多項式が与えられた場合にその値を真にする変数値の組み合わせが存在するかを判定する問題である．SAT 自体は NP（Non-deterministic Polynominal time：非決定性多項式時間）完全問題とされていて，任意の論理式の充足可能性の判定を多項式時間で行うアルゴリズムはないが，実用的な問題に対してはヒューリスティックの工夫により短時間で解くことができる．近年 SAT ソルバの性能が上がってお

り，その能力を利用するというわけである．

NuSMV ではモジュールに状態機械の記述を行う．モジュールには変数や初期状態および状態遷移，制約を記述する．変数にはブール型（boolean）, 整数（int）, 列挙型（enumeration type, ただし記号および整数，その両方を含む型が選択できる），ビット列（word）およびその配列や集合が使用できる．

状態遷移は，次の状態における値を記述する．ASSIGN 文の中で next オペレータを使って記述する方法や TRANS 文を使う方法がある．不変条件も式によっては ASSIGN 文で記述できるが，左辺に 1 変数しか使えない．INVAR 文を使って記述することもできる．

```
module
VAR a : ..; ..

ASSIGN x := TRUE
       init(a) :=
       next(a) :=
```

その上で CTLSPEC（もしくは SPEC）文に仕様としての CTL 式，INVARSPEC に不変条件式を記述し，検査する．

SMV（NuSMV を含む）の適用例は数えきれない．例えば回路設計の事例 [60, 61] や航空機衝突回避システムへの適用事例 [62] などがある．日本国内でも信号保安装置への適用検討例として連動装置への検討事例がある [63]．記号モデル検査を利用することで，状態数に関しては，全てを明示的に列挙するよりも多くの状態に対応できると考えられるが，この事例では 8 進路程度であっても，そのままでは処理しきれない事例があることが報告されている．そこでモデルを構成する部品に着目し，それに関係する部品を残すことで状態を削減することを提案し，検査可能としている．

3.4.2 SPIN/Promela

SPIN もモデル検査法としては草分け的な存在である．1980 年に AT&T のベル研で並行プロセスのモデル化および検査手法として開発が始まり，現在もバージョンアップが続けられている．Promela（Process Meta Language）と呼ばれる言語でモデルを記述する．

SMV と異なり，SPIN/Promela では無限長の語に拡張された有限状態オートマトン（ω オートマトン）の 1 つである Büchi オートマトンをその基礎においている．システムの Büchi オートマトン M_A とシステムが満たすべき仕様の否定の Büchi オートマトン $\overline{M_f}$ に対し，その積が空であれば，仕様が満たされることになるのでそれを判定する，というのが実際に SPIN/Promela が行う検査である．ここでシステムが満たすべき仕様については線形時相論理（LTL : Linear Temporal Logic）での記述が可能となっている．LTL で記述された命題であれば，その否定形の Büchi オートマトンに変換可能であることから，計算量を別とすれば LTL 式に関してその成否を検査できるというのが，理論的背景にある．

Promela では，システムを大域変数，プロセステンプレートおよび通信チャネルによって記述す

る．各プロセスには局所変数も定義可能である．

変数の型としては bit 型，byte 型, short 型, int 型のほか，定数型となる mtype が使用できる．なお，配列も使用できるが配列の値を同時に代入する方法は提供されていない．

プロセステンプレート内には，大域変数や局所変数の値の変更のほか，ガードコマンドが記述される．ガードコマンドは 1 回だけ行う if 文

```
if
  :: ガード条件 1 -> 状態遷移
  :: ガード条件 2 -> 状態遷移
fi
```

と繰り返し行われる do 文

```
do
  :: ガード条件 1 -> 状態遷移
  :: ガード条件 2 -> 状態遷移
od
```

があり，いずれもガード条件が成立すると，以降の状態遷移に制御が移る．複数のガード条件が成立する場合は，そのうちの 1 つが非決定的に選択され，実行される．1 つも成立しない場合はどれか 1 つが実行可能になるまで，実行が停止する．なお，全てのガード条件が成立しない場合に相当する else を記述することが可能である．

このようなガード条件と状態遷移からなるプロセスを複数記述し，それぞれを（状態空間上で）並列に実行させることができる．それぞれのプロセスでは変数を共有することができるので，あるプロセスにおいてガードが成立せずに実行が停止しても，他のプロセスによって状態が遷移することでガード条件が成立するような場合もありうる．このような形で複数のプロセスがお互いに影響しあいながら実行される．

また，チャネルには非同期のバッファ通信とランデブ型の同期通信がある．非同期の場合には送信側はバッファにメッセージを送れば次の文に制御が移る．受信側はバッファにメッセージがあれば次の文に制御が移るが，メッセージがなければ待機状態となる．同期通信の場合には送受信とも通信できる状態になって初めて実行され，次の状態に移る．

検査方法は多様なものが用意されており，

- 特定の箇所にラベル付けをして，その箇所を通過する・しないによりシステムが想定した・していないということを検査する．
- assert 文により，挿入箇所において与えられた条件が満たされているかどうかを検査する．なお，検査用のプロセスを定義し，そこに不変条件を assert 文を用いて記述することにより不変条件の検査も可能となる．
- LTL 式を記述し，その否定に対して never オートマトンを合成して，その LTL 式を満たす状態に遷移するかどうかを検査する．

SPIN 自体はこれらをシミュレーションにより検査するが，これを網羅的に検査するために検査器の

コードを生成して，それをコンパイルしてモデル検査を行うという手続きを採っている．

SPIN の適用例についても数えきれない．SPIN のホームページに挙げられているものだけでも，NASA における宇宙船関連のソフトウェアの解析 [64] やトヨタ自動車の急加速問題に関するプログラム解析 [65] などがある．

信号保安装置への適用に関しては連動装置への適用の検討事例 [66] がある．この事例では標準結線図 [67] を直接 Promela で記述することで検証を実施している．しかしながら当時のメモリが現在よりも小さいとはいうものの，8 進路であっても状態数が多すぎてメモリを使い切ってしまうことが報告されている．連動装置では進路が数十〜数百となることは珍しくなく，本来検証したいこのような事例で，かえって適用できないことになる．そこで文献 [66] では，ある進路に着目した場合にその進路と競合関係にある進路だけを抽出して調べるという手法を提案している．これでも限界がある可能性を論じており，規模が大きくなる場合には積極的に状態削減法を使用することが必要であることが示唆されている．

3.4.3 抽象化に基づくプログラムのモデル検査

モデル検査法を用いて，ソースコードそのものを検証する試みも最近多い．この場合，ソースコードを自動で（あるいは，部分的に手動介入して）抽象化し，それを検査するという手法が採られる．近年研究が盛んな話題である．例としては Microsoft の SLAM[68] や，Java Pathfinder[69] などがある．こういったツールでは assertion を挿入して，その挿入箇所でその制約が満たされるかどうかを検証できる場合もある．このようなツールでもソフトウェアの品質向上に資すると考える．ただし，形式記述した仕様を形式検証することでソフトウェアの品質向上を行うという目的からすると主題が異なる．また，我々は信号保安装置向けのソースコードを直接書く立場になく，適用事例の検討を評価することが難しいことから本論文では詳細な議論を行わない．

3.4.4 その他の事例

そのほかのツールで最近，話題となっているものに SCADE[70] と呼ばれるものがある．SCADE 自体は開発環境であり，グラフィカルな表現でシステムを記述できるが，実際にその裏で使用される言語は ESTEREL[71] や LUSTRE[72] といった同期型言語である．同期型言語においては同期クロックをトリガとしてその時に得られる処理を行い，次のクロックまでに処理を終える．入力の変化が CPU の処理速度に対して十分遅ければ同時性仮説は妥当であるとされる．

MATLAB/Simulink[73] を使うようなモデル駆動開発にも親和性がよいという宣伝がなされているが，信号システムでは MATLAB/Simulink を使う事例は見られない．

この SCADE では Prover Technology 社の Prover Plugin[74] を用いて SAT ソルバを用いた有界モデル検査を行っている．SPIN や SMV といったモデル検査手法に比べると報告はずっと少なくなるが，それでも航空への適用事例 [75] のほか，鉄道に関しても踏切への適用事例 [76] などが報告されている．国内での検討事例については文献 [77, 78] がある．ATC システムに Atelier B を使用した RATP（Régie Autonome des Transports Parisiens：パリ交通公団）では，SCADE による開発も行っているとされているが，RATP から適用事例の報告は見られず，適用検討中の段階と推定される．

SCADE とは直接関係ないが，最初の鉄道へのフォーマルメソッドの実用例とされているスウェーデンの電子連動への検証の適用 [79] では，SAT ベースのツール NP-Tools を適用している．この NP-Tools 自体が Prover Plugin と関連を持っており，バックエンドとなるツールは同じということになる．SAT ベースの検査はいわゆる有界モデル検査と同様のものとなる．ただし，有界モデル検査では必ずしも全ての検査式等が検査できるかどうかは分からない．

第 4 章

検証対象および手法の選択

第 2 章にて信号保安装置に関する安全性技術，第 3 章にてフォーマルメソッドを概観したが，本論文では以下の章で信号保安装置についてフォーマルメソッドを適用に関する検討を行っていく．

フォーマルメソッドにはプロトタイピングのための手法という側面もあるが，本論文の対象は信号保安装置という高安全システムであるため，仕様の正当性の検証につなげる必要があると著者は考える．ここで定理証明とモデル検査法のどちらを適用したらよいかについては意見が分かれるところであるが，モデル検査法の場合，検証可能な対象が純粋な制御部分に限られてしまう可能性が高い．それに対し，定理証明を用いることで，システムが定められた機能を有していることを数学的に証明することで，システムの安全性を高められるのは非常に重要であると考える．また定理証明を使うことにより，B メソッドなどで検証済みのコードが生成可能という点が魅力的であることから，本論文では定理証明の適用を試みる．

4.1　対象の選択

ここで様々な装置やプログラムのうち，ケーススタディとして検証を実施する対象について検討を行った．

本論文で取り上げる装置の範囲については，2 章の冒頭で紹介した基本的な信号保安装置から，機械部分が多く電力も大きいため電子化にはなじまない転てつ装置や，条件に従って表示する機能が主であり電流も大きいためやはり電子化にはなじまない信号機を除く，閉そく装置，連動装置，ATS，ATC といった範囲の装置とする．信号保安装置にはそのほかに踏切保安装置や運行管理装置がある．踏切保安装置については，やはり高い安全性が要求されるものの，踏切保安装置に適用されている技術は，踏切に障害物が存在することを検知する踏切障害物検知装置を除くと，連動装置や閉そく装置と同様の部分が多い．そのため本論文で取り上げることはしないが，本論文の応用として適用が可能である．また運行管理装置については，安全性に対して直接には影響しないことから本論文では対象としないが，適用を否定するものではない．

本論文の対象とする装置の処理の大部分は，数十〜数百 msec 毎に確定した入力を用いて論理演算を行うものとなる．部分的にはそれに加えて数値計算を行うが，ATS 信号（数十 kHz〜数 MHz）や ATC 信号（数 kHz）のアナログ波形を直接変復調する部分を除くと，周期毎の入出力の変化が定義される誤差よりも小さいとみなせる範囲での計算となる．実際には入力が急激に変化する場合がある

が，その変化部分については，過渡現象として扱われ，誤差そのものは重要とされない．

これらの装置を考えた場合，一般的な組み込み用途のプログラムと同様，オフラインで検証可能な静的な部分と，リアルタイムに状態が変化する動的な部分がある．それぞれについて，適用可能性を探る必要がある．そこで，それぞれの部分についてケーススタディを行うこととする．

静的な部分に対する検証の例題としては，制御の基礎データとなる線区データの定義が整合性を持っているかどうかを検証することとした．これは完全にオフラインで編集するものではあるが，定義関係が複雑さを持っており，これが形式検証できれば，様々なシステムの制御データの検証に応用可能であると判断されるからである．

動的な部分に関しては，連動装置などのように歴史的にリレーで論理が組み立てられてきたものと，最近の ATC や ATS の車上装置などのように，車上で数値計算を行った結果をもとに制御を行うものとに分けられる．しかしながら信号保安装置に関して，on-off 以外のアナログの出力が必要な部分は限られているため，数値計算を行う場合であっても，数値計算部分を切り離し，そこで制御の判断を行うための判定式を構成してしまえば，その後の制御は論理演算結果に従って制御する場合と大きく変わることがない．そこで，動的なものについては，論理回路で制御できるようなもの，つまり基本的にブール型あるいはブール型に変換可能な列挙型の変数で構成できるプログラムをまず対象とする．

このような論理で制御される装置の代表は連動装置であるが，複雑であり，いきなり適用を試みるにはハードルが高い．そこで，連動装置と似た制御構造を持ちながら規模が小さい，単線区間の運転方向を制御する閉そく装置を対象とした．これが検証できれば連動装置などにも応用可能である．

それに収まらない数値計算部分としては，ATC などで停止位置までの距離が与えられた場合に，許容される速度を計算するプログラムを対象とした．ここで速度と位置との関係はブレーキ曲線と呼ばれている．この計算は普通に考えると浮動小数点計算や，数学関数が必要となるものであり，このような計算に対して定理証明が適用可能であることが示されれば，広い範囲の装置に適用可能であることを示せる．

まとめると，本論文の内容は以下のとおりとなる．

- 静的な検証：線区データベースの整合性の検証（第 6 章）
- 動的な検証（ブール型および列挙型変数で記述可能）：単線区間向け自動閉そく装置の検証（第 7 章）
- 動的な部分のうち，論理演算では収まらない数値計算：ブレーキ曲線（第 8 章）

これらのケーススタディにより，多くの信号保安装置のプログラムに対する参照モデルを構築することができる．

しかしながら，まずはフォーマルメソッドに習熟する必要がある．そこでまずはフォーマルメソッドを用いて，システムのモデル化が出来るかどうかを検討した．そこで最初に選択したのは，連動装置のモデル化である（第 5 章）．定理証明による検証では難しいと述べたが，モデル化やアニメーションについてはそこまでの困難さはない．また信号保安装置で GUI を作った時にもっとも絵的に分かりやすいため，例題として選択することとした．

4.2　手法の選択

1990 年代後半，著者の所属する鉄道総研がフォーマルメソッドに着目し始めたころ，最初に接した手法は VDM であった．従って必然的に第 5 章に示す連動装置のモデル化については VDM を選択することとなった．まだ当時はフォーマルメソッドの理解が進んでおらず，第 3 章で説明した様々な手法があることもそれほど分かってはいなかったが，幸いにして VDM は形式仕様記述言語としては平易であり，その開発ツール VDM-Toolbox はコード生成機能を有しているために，有利であった．

また，証明機能によって仕様記述に正当性を与えるための研究が進んでおり，これにより仕様の正当性が保証されれば，コード生成器出力の正当性という問題が残るものの，この問題はコード生成器が枯れることで解決できるだろうという判断ができた．そこで第 6 章においても VDM を選択した．

ただし，第 6 章で使用したものは研究段階のプロトタイプであった．結果的には，証明ツールは実用化されず，継続的に使用することが不可能となった．第 7 章および第 8 章では，手法を選択し直す必要性に迫られた．ここではやはりプログラム生成という特徴に着目して，別のモデル規範型言語である B メソッドに選択し直した．現在では Event-B も選択肢として有力であると考えられるが，検討当時ではツールが未熟で選択できなかった．

以上の経緯から，第 5 章および第 6 章では VDM を，第 7 章および第 8 章では B を使用する．

第 5 章

VDM によるラピッドプロトタイピング — 連動装置

本章では，定理証明の適用を試みる前に，フォーマルメソッドのもう 1 つの側面であるラピッドプロトタイピングの手法について，検討事例を示し，その有効性を議論する．対象は連動装置である．連動装置とは，構内において信号機やポイントを列車の衝突や脱線を起こさないように制御する装置であり，駅構内における運転保安の要である．この装置を VDM-SL によりモデル化し，外部プログラムにて GUI を作成してアニメーションを可能とした．ただし，ここではモデル化を簡略化しており，連動装置そのものの安全性検証を対象にしていないことに注意を要する．

このモデルのオリジナルは文献 [80] の連動シミュレータである．連動装置をモデル化の対象に選択したのは，前章で説明した通り，モデル化に限れば連動装置でも適用可能と考えられたこと，また絵的に映え，見た目に分かりやすいといったことが理由として挙げられる．ただしオリジナルの時点においては，CORBA API が提供されていなかったため，仕様からコード生成器にて生成した C コードに外部コードを加えて作成している．

本章においてはそのモデル化を全面的にやり直した．なお，本章の説明では日本語を使って仕様を記述しているが，実際に VDM 記述の見直しや GUI 作成を実施した 1999 年当時においては，VDM で変数名やフィールド名に日本語が使用できなかった（ただし，文字列としては使用できた）ことから英語（ASCII 文字）によって記述している．なお，付録 A に日本語化した VDM 仕様を示す．

5.1 連動装置の概要

連動装置については第 2 章で簡単に紹介したが，ここではもう少し詳しく解説する．駅構内では列車が進むべき経路となる進路を定めておく．その進路上の転てつ器（ポイントを動かす装置）[*1]が所定の向きに鎖錠（ロック）され，競合する（経路が重複する）進路が設定されておらず，進路上に列車が存在しないことを確認できると，その進路に対応した信号機が進行を現示できる[*2]．そして一旦，進路内に列車が進入すると，列車が進路を進出するまで，転てつ器が転換しないように鎖錠しつ

[*1] 鉄道分野では線路を分岐させる部分の構造（保線部門で管理する部分）を「転てつ器」（ポイント），その転てつ器を動かしたり鎖錠したりする機械（信号部門で管理する部分）を「転てつ機」（転てつ装置，転換鎖錠装置）と呼んでいるが，連動図表の記述ではそれらを併せて「転てつ器」と呼んでいる．本章では用語として「転てつ器」を用いる．

[*2] 信号機が灯などで示す符号を現示と呼ぶので，現示するというのは灯により運転士に指示することを意味する．

づける．ここで進路と転てつ器の間には一定の関係（連鎖）があり，これを保つことを連動と呼ぶが，その制御を行うのが連動装置である．

モデル化に際し，以下の非形式仕様を作成した．実際の連動装置では，運転の効率等を考慮してさらに多くの要件が導入されるが，ここでは，システム仕様を単純化した．

1. 列車の走行経路を進路と称する．
2. 進路上には転てつ器がある．ある進路に対して進路上の転てつ器が所定の向きを向いていて鎖錠されており，競合する進路が構成されていなければ，その進路は構成できる．競合する進路については，転てつ器の向きに違うものがあれば転てつ器の条件で排他制御できるが，向きが正反対の進路などでは転てつ器の条件が一致するので，この場合は鎖錠条件に進路名を入れて制御する．
3. 鎖錠されている転てつ器は転換できない．
4. 1 つの進路に対して 1 つの信号機が対応する．
   実装では物理的制約から複数の進路の現示を 1 つの信号機に集約することがしばしば行われる．この場合，集約された進路のうちの 1 つが構成されると進行が現示される．進路表示機という補助的な表示機を併用することも行われる．
5. 信号機は場内信号機か出発信号機のいずれかとする．
   場内信号機：駅に進入する列車に対して信号を示す信号機
   出発信号機：駅から進出する列車に対して信号を示す信号機
   実際には，そのほかに入換信号機，入換標識等があるが，今回は考えない．
6. 信号機は進行（緑）および停止（赤）の 2 種類の現示を出せる．
   実際にはそれ以外の現示もある．また進行と注意（橙色）で連動の条件が異なる場合がある．
7. 進路てこ（進路要求を行うためのレバー）が反位（進路設定が要求されている）[*3] のときにおいて，進路が構成されていて，関連する軌道回路に列車がいなければ，信号機は進行現示を出すことができる．実際の転てつ器は転換に時間がかかるが，ここでは鎖錠されていなければすぐに転換することができる．
8. 進路てこを反位とすると，関連する転てつ器を所定の向きに転換して鎖錠しようとし，競合する進路を鎖錠しようとする．ただし，関係する転てつ器や進路が鎖錠できなければ進路を構成しないで，要求は取り下げられる．
   実際には通常は進路てこを反位とすると進路が構成されるまで要求し続け，条件が成立すると進路が構成される．
9. 鎖錠された進路は構成できない．
10. てっ査鎖錠：転てつ器に関連する軌道回路上に列車が在線している場合には，その転てつ器は転換できない．
11. 列車進入後，進路てこを復位（定位に戻して進路要求を取り下げる）しても，列車が関連する軌道回路を抜けるまでは関連する転てつ器や競合進路の鎖錠を続ける．

[*3] 鉄道信号では定常の位置にある状態を定位，その逆を反位と呼ぶ．進路設定に関しては，通常は要求していない状態が定位と定められているため，設定を要求する場合が反位となる．

```
types
  信号機の状態 = <停止> | <進行>;
  転てつ器の向き = <定位> | <反位>;
  転てつ器の鎖錠 = <鎖錠> | <解錠>;
  軌道回路の状態 = <在線> | <非在線>;
  進路の状態 = <設定> | <未設定> | <鎖錠中>;

  進路名 = token;
  軌道回路名 = token;
  転てつ器名 = token;

  進路条件:: 転てつ器 : map 転てつ器名 to 転てつ器の向き
             鎖錠進路: set of 進路名
             信号制御: set of 軌道回路名
             進路鎖錠: set of 軌道回路名;

  連動図表 :: 進路 : map 進路名 to 進路条件
              転てつ器: map 転てつ器名 to set of 軌道回路名
              軌道回路: set of 軌道回路名
  inv mk_連動図表(進路, 転てつ器, 軌道回路) ==
    (forall r in set dom 進路 &
      dom 進路(r).転てつ器 subset dom 転てつ器 and
      dom 進路(r).鎖錠進路 subset dom 進路 and
      dom 進路(r).信号制御 subset 軌道回路 and
      dom 進路(r).進路鎖錠 subset 軌道回路) and
    (forall p in set dom 転てつ器 & 転てつ器(p) subset 軌道回路) ;
```

図 5.1 連動装置モデルの型定義

5.2 連動装置の VDM モデル化

この非形式仕様をもとに，以下の節において連動装置を VDM-SL を用いてモデル化し，検証を実施できるようにした．

5.2.1 構成要素のモデル化

まずは図 5.1 に示すように型を定義した．文献 [1] によると型を定義する場合には名詞に着目するとよいとされるのでそれに従う．非形式記述から名詞を抽出すると「進路」「転てつ器」「信号機」が抽出できる．そのほかに，駅構内の在線状態を把握するためのデバイスとして「軌道回路」がある．これは図 2.2 で示したとおり，一定の区間内における列車の有無をレールに流す電流により検知する装置であるが，連動図表では区間そのものについても軌道回路と称している．こういった要素間の関係は「連動図表」と呼ばれる図表に記述される．これらの要素が取りうる状態を考え，信号機の状態などの型を抽出した．「信号機の状態」などの型名の右辺にある< >で囲まれたそれぞれが引用型である．「信号機の状態」は引用型の複合型という表記になる．

```
1   state 連動装置 of
2     連動 : 連動図表
3     軌道回路 : map 軌道回路名 to 軌道回路の状態
4     転てつ器 : map 転てつ器名 to 転てつ器の向き
5     転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
6     信号機 : map 進路名 to 信号機の状態
7     進路設定 : map 進路名 to 進路の状態
8   inv mk_連動装置（連動，軌道回路，転てつ器，転てつ器の鎖錠，信号機，進路設定） ==
9     dom 軌道回路 = 連動.軌道回路 and
10    dom 転てつ器 = dom 連動.転てつ器 and
11    dom 転てつ器の鎖錠 = dom 連動.転てつ器 and
12    dom 信号機 = dom 連動.進路 and
13    dom 進路設定 = dom 連動.進路 and
14    てっ査鎖錠(軌道回路，連動.転てつ器，転てつ器の鎖錠) and
15    forall r in set dom 進路設定 &
16      (信号機(r) = <進行> => 進路設定(r) = <設定>) and
17      (進路設定(r) = <設定> =>
18         進路開通(進路設定,連動.進路(r),転てつ器,転てつ器の鎖錠)) and
19      (信号機(r) = <進行> =>
20         forall tc in set 連動.進路(r).信号制御 & 軌道回路(tc) = <非在線>) and
21      (信号機(r) = <停止> =>
22         exists tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <在線>)))
```

図 5.2　状態変数の定義

信号機や転てつ器，軌道回路などの要素の関係を定義したものが連動図表となるが，連動図表型ではレコード型を使用した．「進路」「転てつ器」「軌道回路」が「連動図表」型における各フィールドとなる．また，map … to は写像型であるから，例えば「進路」においては，「進路名」型の値から，それに対応する「進路条件」型の値を得られることを意味している．ここで，軌道回路名や進路名についてはトークン型とした．

「連動図表」型の定義の不変条件は図 5.1 の 20 行目にある inv 以下になるが，ここでは転てつ器や軌道回路が正しく定義されていることを要求している．なお，不変条件において連動図表が正当かどうかについては条件としていないことに注意が必要である．例えば，転てつ器の向きが同じで列車の進む方向が正反対の進路などは本来は互いに鎖錠される進路として記述される必要があるが，ここでは不変条件とはしていない．ここで，図 5.1 の 22〜25 行目にある dom は写像における定義域（domain）である．また 21 行目の forall は全称限定子である．forall に続く部分は dom 進路に属する変数 r を用いて量化することを宣言している．& 以下が r で記述された論理式である．そして 22〜25 行目の各行にある subset は左辺が右辺の部分集合であることを示している．

連動図表型を定義すると，連動装置を記述できる．図 5.2 に今回記述した連動装置のモデルを示す．1 行目は状態変数の集合を連動装置という名で宣言しており，2 行目以降が実際の状態変数の宣言となる．inv 以下が状態変数間の不変条件である．全ての不変条件を一度に記述すると分量が多いため，部分ごとに bool 型の値を返す function として別の箇所に記述する．例えば「進路が開通している場合に，転てつ器が所定の向きに鎖錠されているか？」という問いは以下のように記述できる．

```
forall r in set dom 進路設定 &
  進路設定 (r) = <設定> =>
    ((forall p in set dom 連動. 進路 (r). 転てつ器 &
        転てつ器の鎖錠 (p) = <鎖錠> and
        転てつ器 (p) = 連動. 進路 (r). 転てつ器) and
     (forall r1 in set 連動. 進路 (r). 鎖錠進路 & 進路設定 (r1) = <鎖錠>))
```

3 行目以下を bool 型を返す function として書き直すと以下のようになる.

```
進路開通 : (map 進路名 to 進路の状態) * 進路条件 *
           (map 転てつ器名 to 転てつ器の向き) * (map 転てつ器名 to 鎖錠)  -> bool
進路開通 (進路状態, 進路成立条件, 転てつ器向き, 転てつ器の鎖錠) ==
    (forall p in set dom 進路成立条件. 転てつ器 &
      転てつ器の鎖錠 (p) = <鎖錠> and
      進路成立条件. 転てつ器 (p) = 転てつ器向き (p)) and
    (forall r1 in set 進路成立条件. 鎖錠進路 & 進路状態 (r1) = <鎖錠中>)
pre dom 進路成立条件. 転てつ器 subset dom 転てつ器向き and
    dom 進路成立条件. 転てつ器 subset dom 転てつ器の鎖錠 and
    進路成立条件. 鎖錠進路 subset dom 進路状態;
```

これを「進路開通 (進路設定, 連動. 進路 (r), 転てつ器, 転てつ器の鎖錠)」という形で呼ぶ.

また, 列車が在線している軌道回路上において関連する転てつ器を鎖錠することをてっ査鎖錠と呼ぶことは非形式的仕様に示したが, これをチェックする function は以下のように記述できる.

```
てっ査鎖錠 : (map 軌道回路名 to 軌道回路の状態) *
             (map 転てつ器名 to set of 軌道回路名) * (map 転てつ器名 to 鎖錠) -> bool
てっ査鎖錠 (軌道回路, 鎖錠条件, 転てつ器の鎖錠) ==
  forall tc in set dom 軌道回路状態, p in set dom 鎖錠条件 &
    (tc in set 鎖錠条件 (p) and 軌道回路 (tc) = <在線>)
      => 転てつ器の鎖錠 (p) = <鎖錠>
pre dom 転てつ器の鎖錠 = dom 鎖錠条件 and
    forall tcs in set rng 鎖錠条件 & tcs subset dom 軌道回路;
```

これを「てっ査鎖錠 (軌道回路, 連動. 転てつ器, 転てつ器の鎖錠)」という形で呼んだ.

また「進行を現示している場合に列車が進路上にいないか」という問いは,

```
信号機 (r) = <進行> =>
  forall tc in set 連動. 進路 (r). 信号制御 & 軌道回路 (tc) = <非在線>
```

と記述できる. こうしてまとめたのが図 5.2 の不変条件である.

```
1    進路設定要求 (r : 進路名)
2    ext rd 連動 : 連動図表
3        rd 軌道回路 : map 軌道回路名 to 軌道回路の状態
4        wr 転てつ器 : map 転てつ器名 to 転てつ器の向き
5        wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
6        wr 信号機 : map 進路名 to 信号機の状態
7        wr 進路設定 : map 進路名 to 進路の状態
8    pre r in set dom 進路設定 and
9        進路設定 (r) = <未設定> and
10       (forall r1 in set 連動. 進路 (r). 鎖錠進路 & 進路設定 (r1) <> <設定>) and
11       (forall p in set dom 連動. 進路 (r). 転てつ器 &
12         転てつ器 (p) = 連動. 進路 (r). 転てつ器 (p) or
13         転てつ器の鎖錠 (p) = <解放>) and
14       (forall tc in set 連動. 進路 (r). 信号制御 & 軌道回路 (tc) = <非在線>)
15   post 進路設定 = 進路設定~ ++ {r |-> <設定>} ++
16                  {r1 |-> <鎖錠> | r1 in set 連動. 進路 (r). 鎖錠進路} and
17        転てつ器の鎖錠 = 転てつ器の鎖錠~ ++
18                          {p |-> <鎖錠> | p in set dom 連動. 進路 (r). 転てつ器} and
19        転てつ器 = 転てつ器~ ++ {p |-> 連動. 進路 (r). 転てつ器 (p) |
20                                  p in set dom 連動. 進路 (r). 転てつ器} and
21        信号機 = 信号機~ ++ {r |-> <進行>};
```

図 5.3　進路要求部分の operation

5.2.2　進路設定や列車移動のモデル化

前節では，連動装置の静的な状態を定義した．本節では，進路を設定したり，列車が移動したりする動的な部分を記述する．進路設定部分の記述を図 5.3 に示す．ここでは VDM の本来のスタイルに則り，implicit な形式で記述した．事前条件では，9 行目（全体での行数）で進路が未設定であることを確認し，10 行目で他の競合する進路が設定されていないことを確認している．11 行目からは転てつ器が所定の向きを向いているか，あるいは解錠されていて転換可能であるかを確認している．最後に列車が在線していないことを確認してようやく条件が整うので，これにより進路設定を受け付ける．

post に続く部分が事後条件である．変数に~がついているが，これは operation 適用前の変数値を表す．値を書き換える状態変数は全て写像型とした．例えば信号機については進路 r のみを進行に変えるため，進路 r とそれに対応する値の対応を ++ 演算子で上書きしている．さらに鎖錠すべき進路については鎖錠状態にしている．転てつ器についても進路に関連する転てつ器について，転換と鎖錠を行っている．

次に列車の移動部分も加える．本来であれば列車を想定し，各列車がどの位置に在線しているかを記述する必要がある．しかし列車の長さにより在線状況の変化も微妙に異なるため，正確に列車の移動を表現するためには距離や列車の長さの情報を含めてモデル化する必要があり，複雑となる．ここでは簡単のため，設定された進路に関連する軌道回路のうちの 1 つの在線状態を変える，という記述とした．連動装置側から見ると，列車が移動した場合の状態変化は軌道回路の状態変化として与えら

```
1   列車移動 l (tc: 軌道回路名)
2   ext rd 連動 : 連動図表
3       rd 転てつ器 : map 転てつ器名 to 転てつ器の向き
4       rd 進路設定 : map 進路名 to 進路の状態
5       wr 信号機 : map 進路名 to 信号機の状態
6       wr 軌道回路 : map 軌道回路名 to 軌道回路の状態
7       wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
8   pre tc in set dom 軌道回路 and
9       軌道回路 (tc) = <非在線> and
10      exists r in set dom 進路設定 &
11        進路設定 (r) = <設定> and tc in set 連動. 進路 (r). 信号制御
12  post 信号機 = 信号機~ ++ {r |-> <停止> | r in set dom 進路設定 &
13                  進路設定 (r) = <設定> and tc in set 連動. 進路 (r). 信号制御} and
14       軌道回路 = 軌道回路~ ++ {tc |-> <在線>} and
15       転てつ器の鎖錠 = 転てつ器の鎖錠 ++ {p |-> <鎖錠> |
16                     p in set dom 転てつ器の鎖錠 & tc in set 連動. 転てつ器 (p)};
```

図 5.4 列車移動部分の記述（在線軌道回路が増える場合）

れるためである．例えば，列車が進んで 1 つの軌道回路が新たに列車検知状態となる場合は図 5.4 のように記述できる．事前条件の exists 以下でそのような進路が存在することを確認している．

事後条件においては，信号機に関する内包的記法を用いて，軌道回路 ID から設定済み進路を検索し，その進路を停止現示に変更している．さらに在線状態およびてっ査鎖錠による転てつ器の鎖錠を行っている．

一方，1 つの軌道回路の検知状態が非在線となる場合はもう少し複雑になる．単に進路内の 1 軌道回路の状態変化のみであれば，軌道回路の状態やてっ査鎖錠の状態を変更すればよいが，さらに列車が進路から不在になる場合には，進路要求を取り下げている場合に進路を解錠する必要があるからである．ここでは事後条件において不変条件を満たしていればよいという考えに基づき，図 5.5 に示すように記述した．10 行目からの事後条件部分について，説明をすると以下の通りとなる．12,13 行目において，未設定の進路や，進行を現示している進路はそのままとする．14〜16 行目においては，設定済みの進路について，全ての軌道回路で非在線になれば解錠する．17〜21 行目においては，鎖錠中の進路について，事後においても設定されている進路により鎖錠されていれば鎖錠を続ける．最後の 6 行（22〜27 行目）は転てつ器が進路や軌道回路の状態により鎖錠の状態を決定することを意味している．ここでは全ての進路について，事前事後条件の関係を調べている．

5.3 GUI の作成

VDMTools ではラピッドプロトタイピングを行う上で，外部モジュールとやりとりするためのインタフェースが定められている．

1 つが Dynamic Link 機能である．これは VDM 仕様を外部の C++ プログラムにより実装するもので，これにより VDM の記述では対応しきれない数学関数を実装したり，implicit function の実装を C++ 言語で記述して function を実行させたりすることが可能となる．

もう 1 つが CORBA に基づくものである．CORBA とは分散オブジェクト技術の一つであり，

```
1   列車移動２(tc : 軌道回路名)
2   ext rd 連動 : 連動図表
3       wr 軌道回路 : map 軌道回路名 to 軌道回路の状態
4       rd 信号機 : map 進路名 to 信号機の状態
5       rd 転てつ器 : map 転てつ器名 to 転てつ器の向き
6       wr 進路設定 : map 進路名 to 進路の状態
7       wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
8   pre tc in set dom 軌道回路 and
9       軌道回路(tc) = <在線>
10  post 軌道回路 = 軌道回路~ ++ {tc |-> <非在線>} and
11       (forall r in set dom 進路設定~ &
12         (進路設定~(r) = <未設定> => 進路設定~(r) = 進路設定(r)) and
13         (信号機(r) = <進行> => 進路設定(r) = 進路設定~(r)) and
14         ((信号機(r) = <停止> and 進路設定~(r) = <設定> and
15           forall tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <非在線>)
16           => 進路設定(r) = <未設定>) and
17         ((信号機(r) = <停止> and 進路設定~(r) = <鎖錠中>) =>
18           進路設定(r) =
19             if exists r1 in set dom 進路設定 &
20                進路設定(r1) = <設定> and r1 in set 連動.進路(r).進路鎖錠
21             then <鎖錠中> else <未設定>)) and
22       (forall p in set dom 転てつ器の鎖錠 &
23         if (forall tc in set 連動.転てつ器(p) & 軌道回路(tc) = <非在線>) and
24            (forall r in set dom 進路設定 &
25             進路設定(r) <> <設定> or p not in set dom 連動.進路(r).転てつ器)
26         then 転てつ器の鎖錠(p) = <解放>
27         else 転てつ器の鎖錠(p) = <鎖錠>);
```

図 5.5　列車移動部分の記述（在線軌道回路が減る場合）

プログラミング言語非依存，プラットフォーム非依存，ネットワークプロトコル非依存でソフトウェアコンポーネントの相互利用を可能とする技術である．これにより外部のオブジェクトから VDMTools に対してアクセスが可能となる．この場合に VDMTools はオブジェクトとして見える．

CORBA のオブジェクトはインタフェース記述言語（IDL:Interface Definition Language）を使って，そのインタフェースを記述する．これらは IDL コンパイラによって各言語へマッピングされる．すなわち各言語のソースコードを生成することができる．実際にネットワーク間，プロセス間でデータをやり取りするのは ORB（Object Request Broker）である．ORB ではオブジェクトのデータ構造を変換した上で通信する．Java の場合には JDK（Java Development Kit）に含まれており，C++ の場合にはフリーソフトの omniORB などを使用する．いずれもそれぞれの言語からは，IDL に記述されたインタフェースに従って ORB をオブジェクトとしてアクセスするようになっている．そこで定義されたインタフェースに従ってクライアントプログラムを記述すればよい．

VDMTools に関しては `corba_api.idl` と `metaiv_idl` という IDL ファイルから生成されたパッケージが準備されており，実際に記述が必要なのはクライアントプログラムのみである．VDMTools のマニュアルでは C++, Java のクライアントプログラムの記述例が提供されており，それを編集することで，アクセス部分を記述できる．

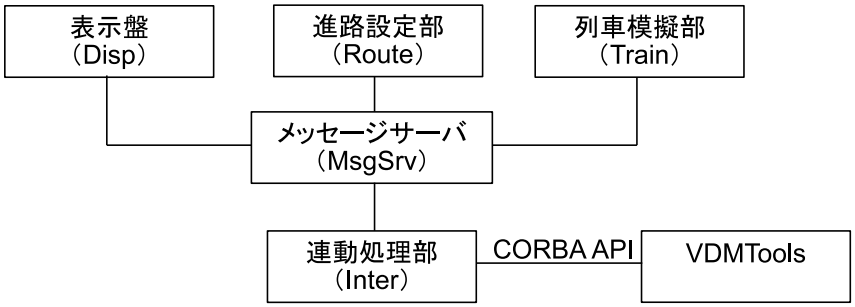


図 5.6 連動装置シミュレータの構成

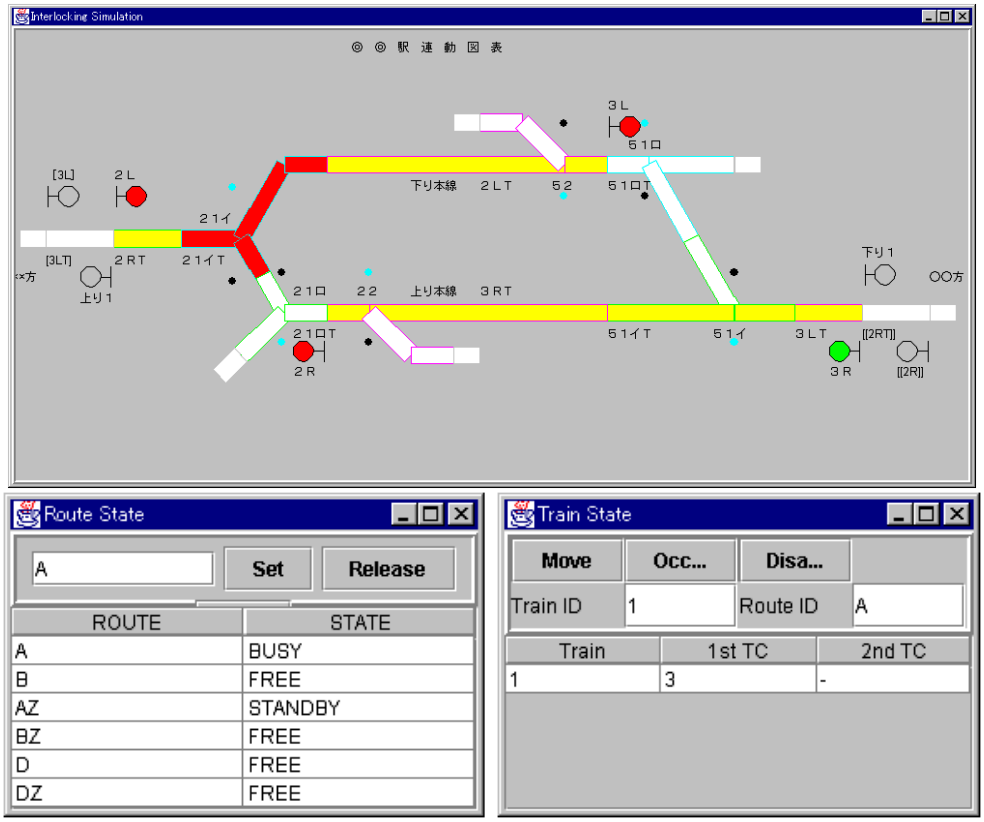


図 5.7 連動装置シミュレータのパネルの表示例

この CORBA API を使い，Java を用いて GUI（Graphical User Interface）を有する連動装置のシミュレータを実装した．シミュレータの構成を図 5.6 に示す．メッセージサーバ（MsgSrv），表示盤（Disp），進路設定部（Route），連動処理部（Inter），列車模擬部（Train）が独立したスレッドを有し，各スレッドがソケットを用いてメッセージを交換する構成としている．表示部を図 5.7 に示す．

このインタフェースを通じて，VDMTools 上で連動装置の VDM モデルを実行し，内部状態を受けとってパネルに表示することにより，インタプリタでは文字情報としてのみ得られる内部情報を分かりやすくグラフィカルに表示できる．これにより仕様が妥当性を有するかの検討を仕様の初期段階で確認できる．

プロセス間で交換するメッセージの内容は，以下の通りとなっている．

```
public char event; イベント識別子
public byte orig;  送信元スレッド
public byte dest;  送信先スレッド
public short len;  メッセージの長さ
public short time_id;  時間 ID
public short request_id;  リクエスト ID
public short n_data;
public short n_size;
```

VDM モデルでは軌道回路名，転てつ器名，進路名はトークン型としたが，Java プログラムではトークンをそのまま扱うことはできないため，String 型配列にその名前を格納することとした．また別途その名前を検索するメソッドを設けた．シミュレータに用いる軌道回路名，転てつ器名，進路名はプログラムに埋め込まず，テーブルを別ファイルに準備した．またパネルについても，進路や軌道回路に対応する要素の位置を別ファイルで用意し，連動部分が変わってもパネル部分のデータを変えることで対応できるようにした．

列車については VDM モデルにはないので模擬部で列車を模擬する．図 5.7 の右下のパネルは列車の模擬部であるが，進入可となっている進路の ID とそれに対応する列車の ID を指定して Occur を押すと，進路の最初の軌道回路に在線する列車を発生させることができる．その後は Move ボタンを押すごとに，列車が 1 軌道回路のみであれば，次の軌道回路を在線状態にし，2 軌道回路にまたがっていれば，後ろの軌道回路を非在線にする．いわば尺取り虫のような動きを模擬している．最終の軌道回路のみに在線しているときに Disappear を押すと列車が駅から進出して消えるようにした．

進路は図 5.7 左下のパネルで操作する．進路名を指定して設定や解除をすればよい．

GUI が VDMTools と直接やりとりする部分は連動処理部に集約した．この連動処理部では，それぞれのスレッドから受け取ったメッセージを解釈し，必要に応じて VDM の値に変換して VDMTools のクライアントオブジェクトを通じて VDMTools にアクセスする．

まずはコネクションを確立する部分を記述する．図 5.8 に示す．まずは 2 行目で VDMTools へのハンドラを獲得する．その後 6,7 行目でプロジェクトの初期化を行い，interlocke.vdm をプロジェクトに追加して，文法チェック，型チェックを行う．それが終わるとインタプリタへのハンドラを取得し，初期化を行う．`DynPreCheck` は事前条件のチェック，`DynInvCheck` は不変条件のチェックのオプションの変更であり，`interp.Initialize()` がインタプリタの初期化である．

インタプリタへのアクセスには 2 つある．1 つめはインタプリタに文字列を送るものである．

```
VDMgeneric g = interp.EvalExpression(client, "route_state");
```

とすると，`route_state` をインタプリタに送って，その結果を g に格納する．g は各型の Helper を使うことにより値を得ることができる．例えば上の例では g は写像であるから

```
VDMMap m = VDMMapHelpler.narrow(g)
```

とすることにより写像として得られる．写像では domain と range の 2 つの集合を持っているが，集合へのアクセスは `VDMgenericHolder` を補助的に使う．例えば

```
1    VDMInterpreter interp;
2    VDMapplication app;
3    short client;
4
5      try {
6       app = (new ToolboxClient ()).getVDMApplication(args,
7              ToolType.SL_TOOLBOX);
8       client = app.Register();
9       try {
10        VDMProject prj = app.GetProject();
11        prj.New();
12
13        prj.AddFile("c:/usr/share/inter_sim/interlocke.vdm");
14        VDMParser parser = app.GetParser();
15        parser.Parse("c:/usr/share/inter_sim/interlocke.vdm");
16        VDMTypeChecker tchk = app.GetTypeChecker();
17        ModuleListHolder moduleholder = new ModuleListHolder();
18        prj.GetModules(moduleholder);
19        String modules[] = moduleholder.value;
20        tchk.TypeCheckList(modules);
21        System.out.println("type check OK");
22
23        interp = app.GetInterpreter();
24        interp.DynPreCheck(true);
25        interp.Verbose(false);
26        interp.DynInvCheck(true);
27        interp.DynTypeCheck(false);
28
29        VDMGeneric gg;
30        VDMRecord r;
31        String str;
32        interp.Initialize();
33        System.out.println("initialized");
34        app.PushTag(client);
35      }
36   }
```

図 5.8 クライアントの記述（VDM とのコネクション確立部分）

```
VDMSet dom = m.Dom();
for(int i = dom.First(eholder); i != 0; i = dom.Next(eholder)){ ... }
```

とすると eholder に domain の 1 つの値が格納される．eholder.value を Helper の narrow メソッドに渡せば，VDM クラスのオブジェクトが得られるし，

```
e.holder.value.ToAscii()
```

とすれば文字列が得られる．

もう 1 つは interpreter クラスの Apply メソッドを用いる方法である．こちらはコマンド名と引数を渡す．引数は Sequence 型の値に格納する．その前に VDMFactory クラスのオブジェクトを宣言することで，引数を構成できる．以下がその例である．arg_l が引数を保持する変数である．

```
VDMFactory fact = app.GetVDMFactory();
Sequence arg_l = fact.mkSequence(client);
arg_l.ImpAppend(g);
arg_l.ImpAppend(Sequence arg_l = argStationAndRouteID(i);
gg = interp.Apply(client,command, arg_l);
```

いずれの方法でもクライアントプログラムから VDMTools にコマンドと値を送ることで，VDM-Tools のインタプリタが処理を実行する．この例では連動処理は VDMTools のインタプリタで実行されており，外部プログラムでは実行していないところがポイントである．形式的に検証する部分は VDM の枠内で実行されることになる．実行結果については，再び連動処理部が VDMTools から状態変数の状態を得て，表示を更新する．

なお，VDM 記述部分は前節に記述したような implicit operation の定義では実行ができないので，別途 explicit operation を定義して，それを実行するようにした．付録 A の operation のうち，列車移動１１および列車移動２１，進路設定要求１１がそれに相当する．事前事後条件はそのまま生かせば，この条件が成立していることに対するチェックが可能となる．

完成した GUI は以下の手順を踏むことにより起動することができる．

1. VDMTools のコンソール版を起動する．
2. GUI を起動する．

   ```
   java -jar Inter.jar  MsgSrv
   ```

不変条件が守られるかどうかをチェックするようにすれば，不変条件が守られない場合にエラーが発生する．例えば，競合する進路が存在しないという不変条件があれば，競合する進路が設定できてしまうと，エラーが発生する．これは仕様の誤りであり，このようなことが発生しないように，仕様を修正する．

エラーメッセージをクライアント側で処理することも可能であるが，VDMTools のコンソール版を使えば，標準出力に出すことができる．

GUI から与える命令が誤っていたり，VDMTools による処理結果に対して GUI での処理が誤っていたりする可能性もあるが，この部分に問題がなければ仕様に不備があることを示しているから，仕様記述を直していけばよい．

5.4　考察および評価

本章では，フォーマルメソッドによる Light Weight なアプローチの事例として VDM による連動装置のモデル化を実施した．ここで GUI を作ってアニメーションを実行することで，モデルの動作状況を簡単に見ることができるようになった．VDM で記述されていない列車模擬部の誤り等もあったが，例えば列車が非在線になっても進路の状態が変わらない誤りを見つけることができた．

このような GUI の利点として，ユーザに VDM の値を直接見せずにアニメーションを実行できることが挙げられる．また，仕様を策定する側から見た場合であっても，インタプリタやスクリプトで実行した場合には状態変数の表示は整形されていない形であるものが，GUI では整理された形で提示されるため有効であると考える．また，このモデルの場合，インタプリタを使おうとすると入力値を構成するのは時間がかかる．その代わりに，GUI で入力を与えることができるため，テストをする上で，扱いやすい方法だといえる．

なお，CORBA API による外部プログラムとの連携が行われる前は，このようなアニメーションを実行するためにはコード生成器出力の活用という手段のみであったが [80]，VDM 部分のみの変更であってもコード生成器による再出力と再コンパイルを必要とするため，手間がかかった．またコード生成器による評価では，ツールのデバッグ機能や評価済みパスのチェック機能が働かないという難点もある．この点からも，CORBA API による外部モジュールとの連携によってテストが実行しやすくなったと言える．

ただし，このような GUI を動かす場合，GUI を作る部分に手間がかかる．従って，手間に見合う結果が得られるかどうかを考慮する必要がある．例えばテストスクリプトを作って実行する場合には，内容によっては CORBA API にこだわる必要はなく，このようなプログラムの必要性も低くなる．

今回の連動装置の例では，データを変更すれば別の駅の装置を試すことも可能であり，GUI の作成は有効であると考えられる．アプリケーションによっては，このようにアニメーションを実行する GUI の製作に時間をかけても十分に行う価値はあると考える．

一方，アニメーションによる検証は，あくまでもテストベースの方法でしかない．妥当性の検証や誤りを早い段階で見つけるためには有効であるが，厳密に仕様が要求を満たしているどうかを保証できるかは別である．よって厳密な検証を実施するには証明という手段が有効になると考える．次章以降では，証明を使って仕様の正当性の検証を実施し，コード生成を行うことを考えていく．

第 6 章

静的なデータ構造への定理証明技術の適用 — ディジタル ATC 線区データベース

前章ではアニメーションを用いた仕様検証の議論を行ったが，本章からは自動証明を用いて仕様の整合性を検証することの有効性の議論を行う．まず本章では前章と同じ VDM を用いた事例で議論を行う．ここで用いたのは，VDMTools の前身である VDM-Toolbox の拡張機能として開発が行われていた証明機能である．

2000 年当時，VDM-Toolbox の開発およびサポートを行っていた IFAD では，EU ESPRIT プロジェクトの 1 つである PROSPER プロジェクト [81] において VDM ツールに証明機能を持たせるための開発を行っていた．この機能は前述した Isabelle での証明機能の変換の流れをくんでいるが，この機能でバックエンドとした証明支援系は Isabelle ではなく，Gordon らが開発した the HOL system[2] （本章では以下特記のない限り HOL は the HOL system を指す）である．

具体的には，VDM-SL の仕様およびその仕様の整合性を示す条件（証明責務）を HOL（HOL98）に変換し，別途作成した GUI を通じて HOL による証明を行う．結果的には，サポートの問題もあり，この機能は実用化されず，それ以降も VDM において証明を行う試みはたびたび行われているものの，研究段階にとどまっている．しかしこの例を通じて，証明責務をレビューすることの意義，自動証明の意義について得た知見は現在でも有効であると考える．証明責務の生成機能については，研究当初の頃の製品では提供されていなかったが，証明責務のレビューを行うだけでも仕様の高信頼化には有効と認められ，その後実装された．

ここでまず検証の対象としたのは，第 4 章で述べた通り静的なデータ構造であり，具体的にはディジタル ATC と呼ばれるシステムの線区データベースである．データベースそのものはリアルタイムに更新されるものではない．しかし，その整合性を検証することには一定の意義がある．これにより制御データの検証が可能であることを示す．まずは線区データベースの紹介を行い，VDM モデル化について説明する．その後，証明責務の生成，自動証明，対話的証明といった手順についてその解説を行い，検証結果に関する考察を行う．

なお，実際に検証を実施した仕様記述について，最終的なものを付録 B に示した．このうち，形式仕様は付録 B.2 となるが，線区データベース仕様を記述した当時は VDM の日本語化が始まったばかりということもあり，コメントを除き，全て ASCII 文字で記述されている．ただ HOL への変換等を考えると現在同じことをしようとしても ASCII 文字で記述を通さざるを得ないと考えられる．

6.1　ディジタル ATC 線区データベースの概要

6.1.1　ディジタル ATC

まずはじめにディジタル ATC について紹介する．

新幹線などで使用されてきた従来の ATC（アナログ ATC と呼ばれる）では，列車は各閉そく区間における最大の許容速度を速度信号（具体的には周波数信号）の形で指示される．このシステムは長年使用され，安全性に関する実績はあげてきたが，列車運行上の効率の観点からは課題も存在する．許容速度の計算はあらかじめ決められた閉そくの配置に従って，ブレーキ性能の悪い列車に合わせて計算され，これがすべての列車に適用される．そのため，減速性能の良い列車については各閉そく区間ごとにブレーキ制御と緩解を繰り返す多段ブレーキ制御（図 6.1 の従来 ATC の走行曲線が示すように曲線が何回も折れ曲がり，階段状となるような制御）となり，必要以上に減速することになるため運転時間が延びたり列車間隔が大きくなったりする原因となる．

それに対する改良がディジタル ATC[82] と呼ばれるものである．このディジタル ATC においては地上から車上への伝送をディジタル化して情報量を大きくした上で，先行列車あるいは駅までの開通区間数[*1]を，列車が在線する軌道回路の ID とともに送信する．車両は受信した情報から，停止位置までの距離およびそこまでの勾配，速度制限といった情報について，車上に搭載されたデータベースを検索する．そして，その車両の減速特性に基づき，位置対速度の関係で表わされるブレーキ曲線を計算することで，停止位置まで車両毎に最適化された一段ブレーキ制御（曲線が折れ曲がることなく，スムーズな制御となっている）により停止する．ここでの本質的な違いは停止までのブレーキ曲線の計算の主体を地上から車上へ変更していることにある．それまでの許容速度の計算は装置を設置する時にあたってあらかじめ行なわれ，地上で速度を決定して指示するため地上主体型 ATC とも呼ばれる．ディジタル ATC では，車上においてその場その場で許容速度を計算するため車上主体型 ATC とも呼ばれる．これにより駅間の運転時間や列車間隔の短縮が期待できる．

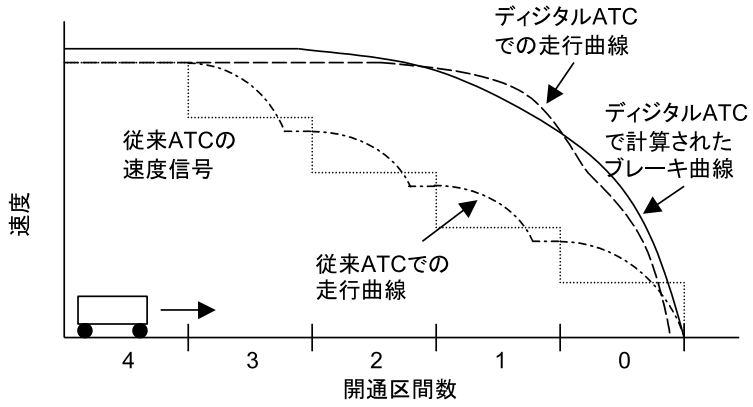


図 6.1　ディジタル ATC システム

[*1] 進路や軌道回路などが列車が進んでよい状態になっていることを開通していると呼ぶ．開通区間数とは列車前方の閉そくがどれだけ開通しているかを示し，列車がどれだけ進んでよいかを示したものとなる．

この概念に基づく車上主体型 ATC は 2002 年 12 月開業の東北新幹線盛岡～八戸間で導入され，現在では更新時期の関係で従来システムが残っている山陽新幹線を除くほぼ全線の新幹線が車上主体型 ATC となった[*2]．結果的にはこれから説明するデータベースが採用されたわけではないが，静的データに関して検証ができることを示すことに一定の意義があると考える．

6.1.2　線区データベースの構造

ATC が車上主体型となることにより，車上の制御装置に搭載されるソフトウェアが，安全性に大きく関わることになる．データベースは静的なものであるが，このデータベースは計算のよりどころであり，高い安全性が求められる．このデータベース構造について，フォーマルメソッドを使って作成を試みた．

車上に搭載される線区データベースでは線路の配置，信号設備，制限速度などが盛り込まれる．具体的な線区データ構造のイメージを図 6.2 に示す．基本的な構想は文献 [83] に従っているが，構造の細部についてはオリジナルとは差異がある．

このデータベースの基本単位となるのは軌道回路（Track Circuit）である．軌道回路とは第 2 章で説明した通り，列車の在線を検知する装置をいうのが本来であるが，第 5 章と同じくここでもその軌道回路で列車の有無を検知できる領域自体を軌道回路と呼ぶ．この軌道回路の情報には信号の搬送波（ATC 信号および無絶縁軌道回路における列車検知（TD: Train Detection）波[*3]）の情報，位置，軌道回路境界の特性などが含まれている．それぞれ軌道回路には一意となる ID を付与する．そして隣接軌道回路との信号および境界の特性には，例えば隣接軌道回路間では信号の周波数が違うといった形での整合性が要求される．

分岐を含まない軌道回路だけで線区が構成できるなら隣接軌道回路の処理は比較的単純であるが，

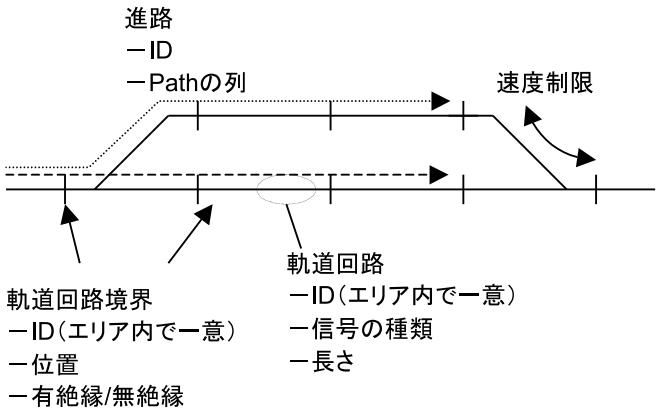


図 6.2　線区データベースの基本構造（エリアレベル）

[*2] 実際には JR 東日本の DS-ATC では，自列車位置と停止位置が与えられたときのブレーキ曲線そのものをあらかじめ計算で求めてデータベース化している．伝送する情報も自列車位置と停止位置で与えており，ブレーキ曲線をその場では計算していない．一方，JR 東海の ATC-NS では随時計算を行っている．

[*3] 通常の軌道回路においてはレールを電気的に区切る絶縁が挿入される．これを有絶縁軌道回路と呼ぶが，電気的な工夫により絶縁が挿入されていないものがあり，これを無絶縁軌道回路と呼ぶ．この場合，ATC 信号と列車検知用の信号は異なるものを用いる．

これは現実的ではなく，実際には分岐を含む軌道回路が多数存在する．この場合は隣接軌道回路は両側に 1 つずつといった単純な話にはならない．そのため，軌道回路内進路（以降 Path と呼ぶ）を定義し，どの境界からどの境界まで走行できるのかを定義する．図 6.3 で言うと境界 1,2,3 に対し，Path は 1 ↔ 2 および 1 ↔ 3 が相当する．分岐器がどういう構造をしているかについてはここでは定義せず，分岐に伴う速度制限をその Path に定義することにする．連動装置とは異なり，線形がどのようになっているかが重要であるからである．Path は分岐のない軌道回路にも定義し，速度制限や勾配等の線区条件は Path 上に設定することにする．

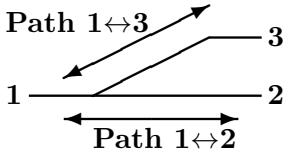


図 6.3　分岐を含む軌道回路と Path

軌道回路および Path の上層に進路（Route）を構築する．進路は Path の順序列で構成され，個々の進路に ID が付与される．列車は常に何らかの進路上を走行することになる[*4]．その進路の ID が列車に送信されることで，列車がこれから進入する軌道回路が特定されることになる．

この軌道回路，Path，進路をエリアと呼ばれる領域にまとめる．各エリアは駅構内あるいは駅中間に対応される．駅中間では単線あるいは複線区間では進路は上下 1 つずつになるだろうから，そのエリア上での進路 ID の伝送を省略できる．さらにエリアを線区にまとめる．ここまでで軌道回路–Path–進路– エリア–線区という階層構造ができ上がる．これらの階層間にはいろいろな制約がある．例えば，ある進路に記述された Path はそれぞれ記述の順に正しくつながっているなど．全体としては複雑な要求になる．それをこれから検証していく．

6.2　線区データベース仕様の VDM モデル化

最初にデータベース仕様を VDM-SL を用いて記述した．

6.2.1　仕様記述上の制約

ここで仕様記述上の制約について述べる．証明機能で実際に扱えるのは VDM-SL のサブセットであり，状態変数や operation には対応していなかった．その他，polymorphism（第 3 章では説明を省略したが，VDM では型変数を function 等の入力で使用できる）にも対応していなかった．対応する VDM のサブセットについては，文献 [84] に詳細がある．VDM 仕様としては function や型定義を中心とした記述になる．また証明機能はに単一モジュールにのみ対応していたが，本章の対象がそもそも単一モジュールを前提としていたため，その点では困難はなかった．

状態変数に相当するものとして，それぞれに対応したフィールドをもつレコード型を定義した．operation は状態変数に対応するレコード型の値を引数とする function として記述した．状態変数

[*4] 本章における進路は，概念的に 5 章の進路と大きく変わるものではないが，駅間にも進路が定められているなど詳細は異なる．

```
TrackC:: joints : map Joint_id to Joint
         atc : map Direction to ATC
         td : TD
         atbt : ATBT
inv tc == card dom tc.joints > 1 and
          dom tc.atc = {<ADIR>, <BDIR>} and
          TD_Used_for_NonInsulated_TrackC(tc.td, tc.atbt, rng tc.joints) and
          (tc.atc(<ADIR>).used and tc.atc(<BDIR>.used) =>
             tc.atc(<ADIR>).carrier <> tc.atc(<BDIR>).carrier)
```

図 6.4 軌道回路型の定義

```
TrackC_map = map TrackC_id to TrackC
inv tcs == forall tid in set dom tcs &
             forall jid in set dom tcs(tid).joints &
               Only_One_Next_TrackC(tcs, tid, jid) and
               forall tid2 in set dom tcs & tid <> tid2 =>
                 Joint_and_Next_TrackC(tcs(tid), tcs(tid2), jid);
```

図 6.5 軌道回路写像型の定義

の書き換えは mutation を用いることで対処できる．例えば

```
x2 = mu(x1, a |-> 'b')
```

とすれば，x1 のうちフィールド a を'b' に書き換えたものを x2 とするという意味なので，x2 を function の返り値とすれば，状態変数 a の値を'b' としたものが function 実行後の状態変数の値に相当すると解釈できる．

function は基本的に explicit に記述した．その理由は，証明責務生成器は implicit function に関する証明責務を生成してくれるものの，satisfiability を示す必要があるというのがその内容だからである．これは事前条件が満たされた場合に，必ず事後条件を満たす解が存在することを証明する必要があるという意味であるが，GUI 上での対応が困難であった．また，satisfiability の証明は，事前条件を満たす入力を用いて，事後条件を満たす値を生成するアルゴリズムを構成すればよいが，これは正に explicit function である．そのためはじめから explicit に記述すればよい．VDM-Toolbox では explicit function に事後条件を付加できる拡張がなされており，explicit function であっても事後条件に関する検証ができるため，問題はない．

6.2.2 エリアレベルの要素の記述

エリアの中で，軌道回路，進路などといったデータベースの構成要素をそれぞれ型として定義していく．これは名詞を抜き出していくという前章と同じアプローチである．例えば軌道回路の構造は軌道回路型（TrackC）として図 6.4 のように定義した．

Joint は軌道回路境界であり，位置と有絶縁／無絶縁の別などを情報として持つ．ATC は ATC 信号の種類，TD は列車検知波の種類，ATBT は信号の送信方向を示している．

VDM では型に不変条件が付加できるので，TrackC 型についても不変条件を付加している．inv

```
1  Path :: tc : TrackC_id
2          start : Joint_id
3          endp : Joint_id
4          length : nat
5          used : map Direction to bool
6          condition : set of Condition
7  inv p == p.start <> p.endp and
8           dom p.used = {<ADIR>, <BDIR>} and
9           (exists dr in set dom p.used & p.used(dr)) and
10          (forall c1, c2 in set p.condition & Condition_not_Conflict(c1, c2));
```

図 6.6　Path 型の定義

```
1  Path_map = map Path_id to Path
2  inv ps == forall pid1, pid2 in set dom ps & pid1 <> pid2 =>
3              (Not_Same_Path(ps(pid1), ps(pid2)) and
4               Not_Start_and_End(ps(pid1), ps(pid2)));
```

図 6.7　Path 写像型の定義

以下が不変条件であり，これは次の意味を持つ．不変条件の 1 行目は軌道回路の隣接軌道回路との境界は 2 つ以上必要である，ということである．2 行目は ATC 信号については A 方向と B 方向について別々の情報を持っているという意味となる．3 行目では bool 値を返す function を述語として別途用いているが，その中身は列車検知波は無絶縁軌道回路で使用されているかどうかをチェックするものとなっている．そして 4,5 行目は両方向で信号の搬送波が異なることを意味している．

軌道回路の集合は図 6.5 に示すように写像型としている．`TrackC_id` はトークン型とした．`TrackC` 型でもそうであるが，bool 値を返す function を述語として使えるように積極的に定義し，さらにその名前には意味を示すようにすることで，仕様としての可読性を上げるように努めた．最後の行に `tid <> tid2` とあるのはその後に適用される function において入力が同じ値となるのをさけるためのものである．

Path については図 6.6 に示す通りとなっている．所属する軌道回路 ID および両端の境界 ID を情報として持っている．さらに情報として長さを持っており，これが距離計算の基礎となる．Path には勾配，曲線，制限速度などの `Condition` を情報として持っているが，それが一意の情報になるように不変条件でチェックしている．

Path の集合も図 6.7 に示すように写像型とした．

進路型は図 6.8 に示すが，ここでは方向と Path の列のみが定義されており，不変条件がない．これは，この情報だけでは Path 同士のつながりが把握できないためである．

軌道回路と Path，進路の間の関係にはそれぞれ単独では記述できないものが多くある．例えば Path は軌道回路境界 2 つを情報として持っているが，その境界が同じ軌道回路に登録されている必要がある．このようなことを記述するにはこれまでの情報を束ねる存在が必要であり，それがエリアとなる．エリアについては，図 6.9 に示す．

エリアの定義では，これまで定義された写像型を用い，`trackcs`，`paths`，`routes` を状態変数相当として考えている．`Area_Kind` はエリアの種類を示している．`<PLAIN>`は駅中間を想定しており，方向と軌道回路に対して，進路が一意に定まるエリア，`<COMPLEX>`は駅構内を想定しており，一意に

```
1  Route :: dr : Direction
2              paths : seq1 of Path_id;
1  Route_map = map Route_id to Route
```

図 6.8 Route 型の定義

```
1  Area :: trackcs : TrackC_map
2          paths : Path_map
3          routes : Route_map
4          kind : Area_Kind
5          max : MaxSpeed
6  inv mk_Area(trackcs, paths, routes, -, -) ==
7      (forall p in set rng paths &
8          Path_within_TrackC(trackcs, p) and
9          Direction_Correct(trackcs, p)) and
10     (forall r in set rng routes &
11         Path_Exists(paths, r.paths, r.dr) and
12         Exists_ATC_for_Route(trackcs, paths, r) and
13         Route_not_Circular(paths, r) and
14         Path_Connected(paths, r.paths, r.dr))
```

```
1    Area_Kind = <PLAIN> | <COMPLEX>;
```

図 6.9 エリアの定義

定まらないエリアである．新幹線では 10km 以上の長さを探索する必要があるが，駅中間と駅構内を分けることにより，駅中間部から駅構内の進路を探索しやすくしている．

エリアに関しても不変条件では述語を多用している．エリアの不変条件に記述された述語は順に以下のとおりとなる．

- Path の両端は同一軌道回路の中に存在する必要がある．（図 6.9 全体（以下同様）の 8 行目）
- Path におけるキロ程（起点からの線路延長）は方向と整合性が取れている必要がある．（9 行目）
- 進路で記述された Path が存在する．（11 行目）
- 進路に対し，ATC 信号が存在する．（12 行目）
- 進路がループとなっていない．（13 行目）
- 進路上の Path が正しい順番で接続されている．（14 行目）

6.2.3 エリアレベルの操作の定義

こうしてデータベース構造をモデル化した後に，最低限のデータベース操作を定義した．この操作に関しても証明機能の制約から function で定義しており，現在の状態と，操作の際に必要な情報を引数とし，新しい状態を返り値として返す．例えば軌道回路を追加登録する操作 `Add_TrackC` は図 6.10 のように定義した．引数 `ar` は現在の状態，`tcid` は新たに追加する軌道回路に付与される ID，`tc` は軌道回路の具体的な情報である．これを関数の内部では軌道回路情報として `tcid` から `tc` が検

```
1    Add_TrackC : Area * TrackC_id * TrackC -> Area
2    Add_TrackC(ar, tcid, tc) ==
3      mu(ar, trackcs |-> ar.trackcs ++ {tcid |-> tc})
4    pre tcid not in set dom ar.trackcs and
5        forall jid in set dom tc.joints &
6          Only_One_Next_TrackC(...) and
7          forall tcid1 in set dom ar.trackcs & Joint_and_Next_TrackC_Consistent(...)
```

図 6.10　軌道回路を追加登録する操作の定義

索できるように登録している．返り値としては軌道回路が追加登録されて更新されたデータとなる．

これだけではどのような軌道回路でも登録できることになるが，それでは矛盾が起こる場合があるため，入力に制限を加える必要がある．これを function の事前条件として定義した．

事前条件の最初の行（図 6.10 全体の 4 行目）では，tcid がまだ登録されていないことを要求している．tcid に関する変更については用意せず，一旦削除した上で再度登録することをここでは想定している．それに続く 2 行（5,6 行目）では，全ての境界に対して隣接軌道回路は 1 つだけである，ということを意味しており，最後の行（7 行目）では隣接軌道回路とは境界と信号について整合が取れているという条件を言っている．これらは述語の真偽値を返す部分を別の function として記述することで対応している．

データベース操作として軌道回路，軌道回路境界，Path，進路，速度制限の追加削除を記述した．

6.2.4　線区レベルの記述

エリアの上位にあるのが線区（line）となる．線区はエリアが接続されて構成されるが，そこでエリア間接続（Connect）という概念を導入した．エリア間接続に関する型の定義を図 6.11 に示す．Area_Joint は，エリアが他のエリアと接続される軌道回路境界を示している．Connect は Area_Joint 2 つにより定義されることで表現されている．さらにその接続に対して，方向やキロ程が不連続[*5]になるかどうかを示したのが Remark_Connect である．

この Connect の集合および，エリアの集合をまとめたのが線区である．線区の定義を図 6.12 に示す．この定義においてもエリアでの定義と同じように，エリアの集合 areas およびエリア間接続の集合 connect を状態変数相当として持っている．前節の定義では Area の下に各項目が状態変数として記述されていたが，ここでは状態変数としてエリアが定義されており，その中に軌道回路や Path などの要素が定義されている．

不変条件は以下の通りとなっている．5 行目（図 6.12 全体の中の行数，以下同様）では，他のエリアとの接続点がエリア内に存在することを意味している．7,8 行目ではエリア間接続につながる Path の方向とエリア間接続における方向の条件の整合性が取れていることを意味している．最後の 2 行では，エリア間接続に示された軌道回路境界情報が整合性を持っていること，またその接続点を挟んだ隣接する軌道回路の信号が整合性を持っていることを意味している．

線区に関しても操作関数を用意した．エリアそのものを追加したり，削除したりする操作，さらに

[*5] 線路を付け替えたりすると，線路延長が変更になるが，その影響が線路全体に及ばないように，必要に応じてキロ程を重複させたり，中断させたりする．

```
1  Area_Joint :: aid : Area_id
2                tcid : TrackC_id
3                no : Joint_id;

1  Connect = set of Area_Joint
2  inv con == card con = 2 and
3             forall a1, a2 in set con & a1 <> a2 => a1.aid <> a2.aid;

1  Connect_map = map Connect to Remark_Connect
2  inv con == forall a1, a2 in set dom con & a1 <> a2 => a1 inter a2 = {};

1  Remark_Connect :: chng_direction : bool
2                    chng_distance : bool;
```

図 6.11 エリア間接続（Connect）の定義

登録されたエリアの中に軌道回路や軌道回路境界，Path，進路を追加削除する操作，そしてエリア間接続を追加削除する操作である．

最後にデータベースが完成したときの条件を作成した．データベースの要素を編集中には成立しない条件だが，編集終了時に成立を求める条件である．図 6.13 にデータベース完成時の条件に対応する VDM 記述を示す．これ自身はやはり述語として記述している．ここでは順番に以下の要件を求めている．

- 軌道回路境界が完全に定義されている．（図 6.13（以下同様）の 4 行目）
- 軌道回路境界に対し，Path が定義されている．（5 行目）
- 軌道回路に対し，Path が定義されている．（6 行目）
- Path に対し，進路が定義されている．（7 行目）
- Path の始点に対し，前に接続される Path が存在するか，あるいは端（車止め，あるいは ATC 制御の始端）として定義されている．（8 行目）
- Path の終点に対し，後ろに接続される Path が存在するか，あるいは端として定義されている．（9 行目）
- 進路を進んでいくとエリア間接続もしくは終端に到達できる．（10 行目）
- 駅中間では軌道回路 ID と方向から進路が一意に定まる．（11 行目）
- エリア間接続箇所では接続箇所を始点とする Path が存在する．（12 行目）
- エリア間接続箇所では接続箇所を終点とする Path が存在する．（13 行目）
- エリア間接続箇所では Path が一意となる．（14 行目）

実際には後に述べる証明作業で誤りや矛盾が見つかれば，その修正も加わるので，すぐに完成とはならないが，最終的には仕様は VDM-SL の記述で 935 行[*6]に達した．

[*6] 当初発表時は 985 行としていたが，これには空行を含む．さらに付録 C に掲載するにあたり，複数行を 1 行にまとめた結果，935 行となっている．

```
Line :: areas : Area_map
        connect : Connect_map
inv mk_Line(areas, connect) ==
    forall c in set dom connect &
      (forall n in set c & Area_Joint_Exists(areas, n)) and
      (forall n1, n2 in set c & n1 <> n2 =>
         Direction_for_Area_Joint(areas(n1.aid).paths, n1.no,
                  areas(n2.aid).paths, n2.no, connect(c).chng_direction) and
         let tc1 = areas(n1.aid).trackcs(n1.tcid),
             tc2 = areas(n2.aid).trackcs(n2.tcid) in
           Joint_Compatible(tc1.joints(n1.no), tc2.joints(n2.no), connect(c)) and
           Is_wf_adjacent_signal(tc1, n1.no, tc2, n2.no, connect(c).chng_direction));
```

図 6.12　線区（Line）の定義

```
Is_wf_Line_DB : Line -> bool
Is_wf_Line_DB(ln) ==
  (forall aid in set dom ln.areas & let ar = ln.areas(aid) in
     Joint_Completed(ar.trackcs, aid, ln.connect) and
     Path_Exists_for_Joint(ar.trackcs, ar.paths) and
     Path_Exists_for_TrackC(ar.trackcs, ar.paths) and
     Route_Exists_for_Path(ar) and
     Path_Exists_before_Start(ar, aid, ln.connect) and
     Path_Exists_after_End(ar, aid, ln.connect) and
     Route_Exists_to_Terminal(ar, aid, ln.connect) and
     (ar.kind = <PLAIN> => Is_Plain_Area(ar, aid, ln.connect))) and

     Following_Path_Exists_at_Connect(ln) and
     Preceding_Path_Exists_at_Connect(ln) and
     One_Side_Unique_Path_at_Connection(ln);
```

図 6.13　データベース完成の条件

6.3　証明責務と証明器

6.3.1　証明責務 (Proof Obligations)

このようにして記述した形式的仕様に文法チェックおよび型チェックをかけて問題が無ければ，証明責務（Proof Obligations）を自動で生成することができる．これは，仕様が矛盾のないことを保証するために必要な条件である．もしこれが一つでも満たされなければ，それは仕様の誤りがあることを意味しており，仕様の修正が必要になる．不変条件として安全性に関する要件が盛り込まれた場合には，その安全性を守れるかが問われることとなる．

証明責務については文献 [21] に function の健全性に関する証明責務が紹介されている．explicit function においては，事前条件が bool 型であること（ただし，これは型チェックでチェックされる），計算結果がその型を満たすことが要求される．また引数については指定された型であることが要求される．ここで不変条件付きの型が使われていれば，ある値がその型に属するためには，その不変条件

が真であることが公理とされるので，結局のところ引数や計算結果について，指定された型の不変条件を満たすことを要求される．

implicit function については事前条件，事後条件が bool 型であること（これも型チェックでチェックされる）のほか，safisfiability，すなわち事前条件を満たす任意の値に対して，事後条件を満たす値が存在することが要求される．ただ，これだけでは十分でない．仕様記述にある演算子（除算や写像の適用なども含む）はたいていが部分関数であるので，被演算子の型が演算子の要求する型に適合し，なおかつその適用条件を満たす必要がある．そのほか，subtype に関する証明責務も生成される．たとえば 1 以上の自然数を示す `NAT1` 型が function の入出力にある場合に，入力変数に `NAT` 型の値を適用する場合などである．

それを踏まえ，型の不変条件および function について，前から評価を行い，演算子や function の適用部分で証明責務を生成していく．後ろの部分はそれまでに評価した部分に関する証明責務が成立していると仮定する．よって，個々の証明責務は，証明すべき位置までに評価した部分をコンテキストとして，それに対して述語が成立すべきという形を取る．実際の証明責務生成器については文献 [85] に解説がある．

function の計算結果に対する不変条件成立の証明責務については仕様の証明として本来行わなければいけない種類のものであり，単純ではない．例として先の `Add_TrackC` に関して生成されたものをあげる．

```
forall ar: Area, tcid: TrackC_id, tc: TrackC &
  pre_Add_TrackC(ar, tcid, tc) => inv_Area(Add_TrackC(ar, tcid, tc))
```

これは任意の `Area` 型の値 `ar`，`TrackC_id` 型の値 `tcid`，`TrackC` 型の値 `tc` の組に対して，関数 `Add_TrackC` の事前条件が満たされていれば，`Add_TrackC` の返り値が `Area` 型の不変条件を満たす必要がある，という意味である．これが真であれば，引数にはそれぞれ指定された型の条件を満たしている任意の値を使用できるため，証明がなされることの意味は大きくなる．これを否定するには `forall` を否定するため，1 つの反例をあげることができれば十分である．

しかし，実際に生成される証明責務の多くは function の事前条件や写像の適用条件である．この場合は注意深く仕様を記述すれば述語をコンテキストの一部に一致させることが可能で，完全に一致する場合は生成器が成立と判断して取り除く．そうでない場合も比較的証明は容易である．例えば以下の function がある．

```
Line_Add_TrackC : Line * Area_id * TrackC_id * TrackC -> Line
Line_Add_TrackC(ln, aid, tcid, tc) ==
  mu(ln, areas |-> ln.areas ++ {aid |-> Add_TrackC(ln.areas(aid), tcid, tc)})
```

この関数中で `Add_TrackC` を呼び出しているが，ここで以下の証明責務が発生する．

```
forall ln : Line, aid : Area_id, tcid : TrackC_id, tc : TrackC &
pre_Line_Add_TrackC(ln, aid, tcid, tc) ==>
  pre_Add_TrackC(ln.areas(aid), tcid, tc)
```

すなわち，`Add_TrackC` の事前条件を満たす必要がある．この場合は `Line_Add_TrackC` の事前条件

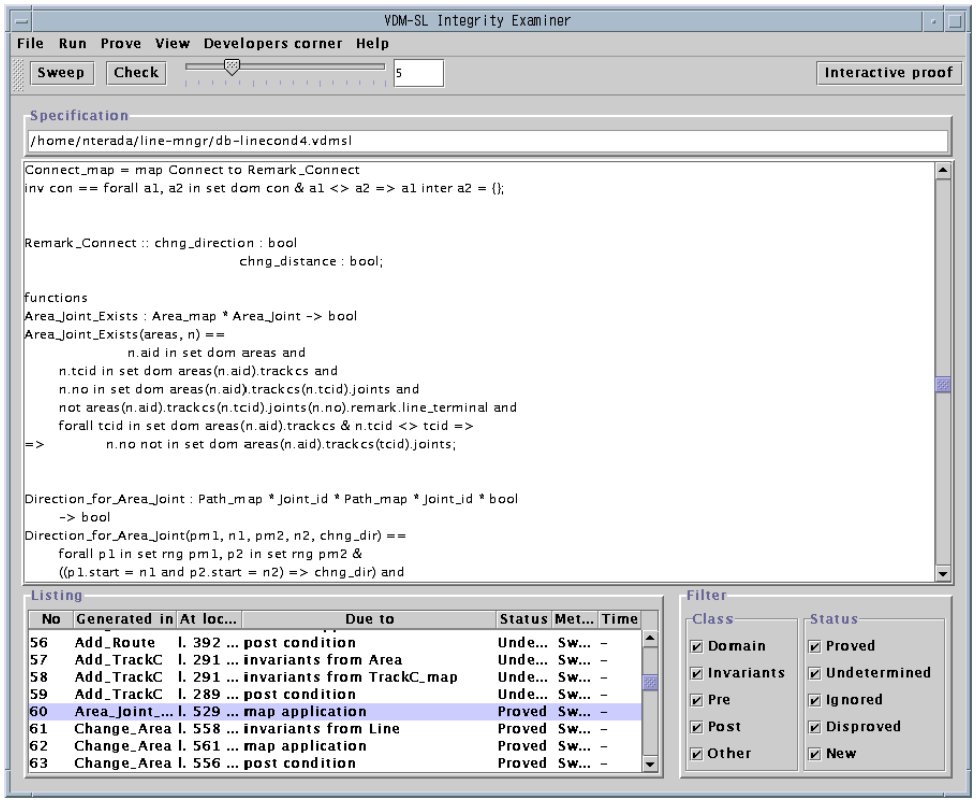


図 6.14　証明用環境

において `Add_TrackC` の事前条件を加えておけばよく，証明は簡単にできる．

935 行の仕様から 188 の証明責務が生成され，VDM-SL で記述すると 2104 行になる．ここでは生成段階で自明である証明責務は除かれているため，本当の証明責務はさらに多くなる．生成された分量を見ても分かるように，証明責務はもとの仕様と比べるとかなり大きくなる．この生成は自動であって，生成規則さえ確立してしまえば取りこぼしはない．今回使用した証明機能では図 6.14 に示される GUI によって生成された証明責務を種類別に分類して眺めたり，証明責務が仕様のどの部分から発生したのかを調べたりすることができる．

証明責務の証明は厳密さを欠くが手作業で行うこともできる．証明を試みた場合に証明できないのは，仕様に誤りや矛盾があり証明が不可能な場合か，証明が難しくて判定できない場合のいずれかである．仕様記述を実施していた当初は次節以降で説明する証明機能が実装されていなかったため，手作業で証明責務が成り立つかどうかを確かめた．その結果，初期段階の仕様について，反例を探す方法でいくつかの誤りを見つけることができた．証明責務の多くは function の引数に関連していることから全称限定子 `forall` で始まる述語となるため，その反例を見つければよい．反例の確認は VDM-Toolbox のインタプリタを使って行った．たとえ手作業で見つけた反例であっても，証明責務が偽であることの確認は機械上でできるためあいまいさはない．

このことから証明責務の自動生成機能は，手作業で証明をする場合であっても非常に強力なことがわかった．もちろん，これが自動証明できれば，もっと効率は良くなるし，さらに大きな仕様では必要不可欠と考えられる．

6.3.2 自動証明

VDM-Toolbox への証明機能の追加は PROSPER プロジェクトの主要成果物である PROSPER Toolkit に基づいている．Gordon らが開発してきた HOL をバックエンドとして使用する．PROSPER Toolkit 自体は他のプログラムとのやりとりのための API 程度しか持っていないが，HOL で開発された公理・定理群および証明ステップ（tactic）が使用可能で，さらに必要に応じてユーザーが定理や証明ステップを追加することができる．VDM-Toolbox 用の証明エンジンはこの PROSPER Toolkit に VDM-SL（具体的には VDM-SL から変換された HOL の仕様）に関する定理や VDM-SL 仕様の証明責務用の自動証明プロセスを追加して作り上げたものである．なお，最終段階の解説文書はないが，開発途中の解説が文献 [86] にある．本章の冒頭で述べたように ISO 標準 VDM-SL のフルセットが使えるわけではないが，使用できる表現に直せることが多いので，ここでは必ずしも問題とはならない，

この証明エンジンの操作については HOL をそれほど詳しく知らなくても証明ができるように GUI が作られており，GUI を通じて証明を行う．GUI の実装は Java で行われ，CORBA API を活用していたが，VDM-Toolbox 自体にも多少の改修を行っている．既出の図 6.14 に自動証明のボタンが用意されている．仕様およびその証明責務はその内部形式である HOL の表現に変換された上で証明エンジンに読み込まれる．この時点で 2 つの自動証明作業が使用可能である．1 つめは **Sweep** と呼ばれ，証明を短時間（5 秒）で強制終了するものであるが，これで証明が単純なものを取り除くことができる．さらに **Check** という作業が用意されている．このタイムアウトはもう少し長いため，多少は深く調べるようになっている．もし証明ができない場合でも証明すべき副命題（subgoal）が生成されて報告される．この残った副命題を全て証明できれば，その証明責務は証明されたことになるが，そうでなくとも仕様の誤りの発見につながることがある．実際にいくつかの誤りがこの段階で発見された．見つかった誤りは修正して，再度証明を行うことになる．最終的には 188 の証明責務のうち 167 が完全に自動で証明された．

6.3.3 対話的証明

残った証明責務に関しては，証明エンジンを使って対話的に証明を行うことになる．対話的証明は Backward Proof のスタイルである．対話的操作画面を図 6.15 に示す．仮定（Assumptions）と結論（Conclusions）を示す窓がある．そのまわりにボタンが並んでいるが，それぞれが証明操作のうちの個々のステップ（tactic）と関係している．例えば，仮定や結論を単純化する，関数の中身を展開するなど．このボタンの操作を組合せていって証明をしていく．

提供されているボタンには以下に示すものがある．結論に対しては，

Strip 全称限定子の除去（つまり限定子を自由変数に変換する）や含意の左辺の仮定への移動，Goal の subgoal への分割等を行う．

Simplify 結論を仮定や書き換え規則にしたがい単純化する．書き換え規則を増やした Simplify（expand in）というものもある．

Contra 背理法

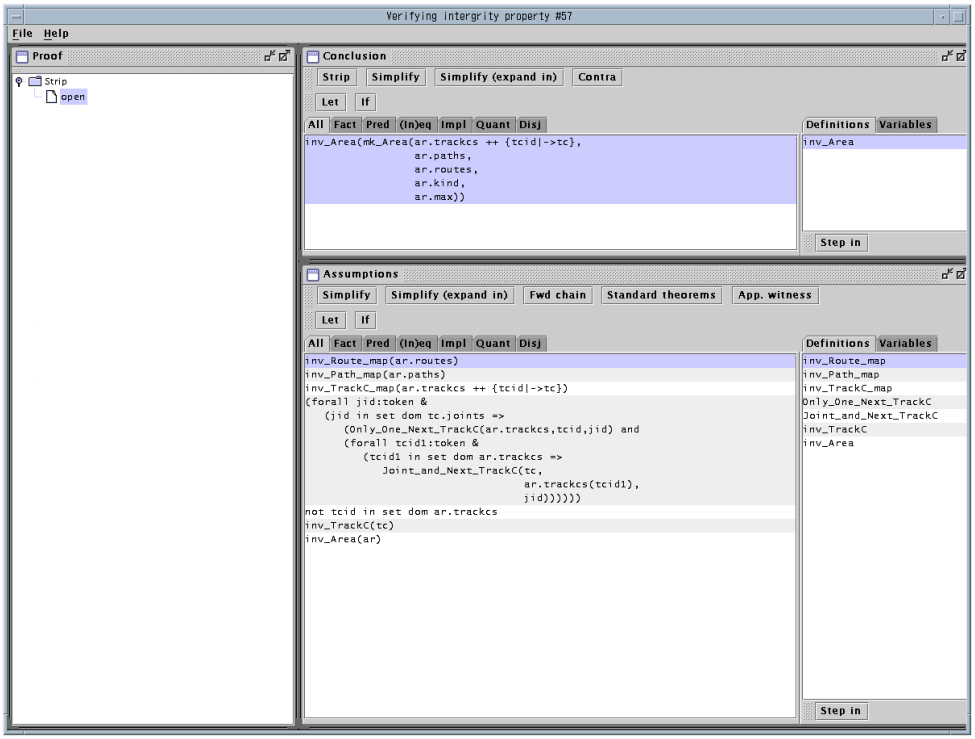


図 6.15　対話的証明画面

Let　let 文の展開

Step In function の定義を展開する．展開する function は Definitions パネルから選択する．

Compare 2 つの変数が等しいか否かで場合分けをする．

Enumerate bool 型か引用型の値で場合分けをする．

仮定に対しては，

Simplify 他の仮定や書き換え規則に従い単純化する．

Fwd. chain `forall` で始まる仮定に適合する変数の組み合わせを代入して，新しい仮定を生成する．

Standard theorems 定理を仮定に適用し，新しい仮定を生成する．

App. witness 写像において range の値に対応する domain の値を新たな変数として導入する．列の要素に対する index にも対応

その他 **Let, Step In, Compare, Enumerate** もある．

なお，補題を追加して証明するようなことはできない．

証明の手順は木構造で保存され，任意の subgoal を選択して証明を進めることができる．

完全な自動証明に比べると対話的証明はユーザにとって難しい作業である．特に証明を進めるためには仕様の構造を良く理解しておく必要がある．本事例では述語を function による記述で分割する手法を多用したため，その定義を展開する必要がある．適切に展開を行うためには，各 function の具体的な定義について把握しておく必要がある．これは仕様の理解にほかならない．この事例の場合，対話的証明は数十ステップから 100 近くまで達した．慣れの程度にもよるが 1 つの証明責務の対

```
1  Add_Path : Area * Path_id * Path -> Area
2  Add_Path(ar, pid, path) ==
3        mu(ar, paths |-> ar.paths ++ {pid |-> path})
4  pre   pid not in set dom ar.paths and
5        Path_within_TrackC(ar.trackcs, path) and
6        Direction_Correct(ar.trackcs, path) and
7        forall p in set rng ar.paths &
8            Not_Same_Path(p, path) and Not_Start_and_End(p, path)
9  post  pid in set dom RESULT.paths and
10       RESULT.paths = ar.paths ++ {pid |-> path} and
11       RESULT.paths(pid) = path and
12       RESULT.trackcs = ar.trackcs and
13       RESULT.routes = ar.routes;
```

図 6.16 Add_Path の定義

話的証明には，不変条件に関しては，数十分から場合によっては数時間を要する．

このような操作を経て，最終的にはすべての証明責務が機械的に証明された．つまり，仕様が整合性をもったものであることが曖昧さのない形で示されたことになる．

6.3.4 実際の対話的証明の実行例

実際の証明の実行例をここで紹介する．ただし，記述を簡単にするため，ここで示される表現と実際の証明に現れる表現の間には若干の違いがあることを断っておく．（証明の概略は変わらない）

ここでの例は図 6.16 に示す Add_Path という function に関するものである．

この function に関して以下の証明責務が生成される．

```
(forall ar : Area, pid : Path_id, path : Path &
    pid not in set dom (ar.paths) and
    Path_within_TrackC(ar.trackcs, path) and
    Direction_Correct(ar.trackcs, path) and
    (forall p in set rng (ar.paths) &
        Not_Same_Path(p, path) and
        Not_Start_and_End(p, path)) =>
    inv_Area(mu(ar,paths|->ar.paths ++ {pid |-> path})))
```

この証明責務は全自動では証明ができなかった．そのため対話的証明機能での処理を開始する．大きく分けて 2 つのウィンドウが現れる．仮定（Assumptions）と結論（Conclusions）である．最初に起動したときには証明責務全体が結論として扱われる．

この例に限らず，最初に行うことはたいてい **Strip** である．この結果は以下のようになる．

```
Conclusions:
  inv_Area(mu(ar, paths |-> ar.paths ++ {pid |-> path}))
Assumptions:
```

```
inv mk_Area(trackcs, paths, routes, -, -) ==
   (forall p in set rng paths &
       Path_within_TrackC(trackcs, p) and
       Direction_Correct(trackcs, p)) and
   (forall r in set rng routes &
       Path_Exists(paths, r.paths, r.dr) and
       Exists_ATC_for_Route(trackcs, paths, r) and
       Route_not_Circular(paths, r) and
       Path_Connected(paths, r.paths, r.dr));
```

図 6.17　Area 型の不変条件（再掲）

```
inv_Area(ar)
inv_Path(path)
pid not in set dom (ar.paths) and
Path_within_TrackC(ar.trackcs, path) and
Direction_Correct(ar.trackcs, path) and
(forall p in set rng (ar.paths) &
    Not_Same_Path(p, path) and
    Not_Start_and_End(p, path))
inv_Path_map(ar.paths ++ {pid |-> path})
```

Strip を適用することにより，まず結論の forall が取り除かれ，その束縛変数だった ar, pid, path が自由変数に変換される．さらに forall の中の ar は Area の型を持っているので，その型の不変条件を満たす必要があり，inv_Area(ar) という条件が仮定に追加される．これは path についても同様で inv_Path(path) が追加される．なお，pid については，不変条件が設定されていないので条件は追加されない．次に証明責務のコンテキストの部分が仮定の部分に移動される．最後に inv_Path_map(...) が inv_Area の中間の条件として追加される．これは，他の証明責務として生成されているため，ここでは証明を省略することができる．

次に **Step In** を適用し，inv_Area の定義を展開する．inv_Area の定義を図 6.17 に再掲する．従って，結論は以下のようになる．

```
Conclusions:
  (forall p in set rng ar.paths &
      Path_within_TrackC(ar.trackcs, p) and
      Direction_Correct(ar.trackcs, p)) and
  (forall r in set rng ar.routes &
      Path_Exists(ar.paths ++ {pid |-> path}, r.paths, r.dr) and
      Exists_ATC_for_Route(ar.trackcs, ar.paths ++ {pid |-> path}, r) and
      Route_not_Circular(ar.paths ++ {pid |-> path}, r) and
      Path_Connected(ar.paths ++ {pid |-> path}, r.paths, r.dr))
```

この結論を見ると大きく 2 つに分けることができる．これは **Strip** を適用することによって分割することができる．この時 **Strip** はさらに `forall` を除去する．

これを整理した後，最初の副命題 (1) は以下のようになる．

```
Subgoal(1):
  Conclusions:
    Path_within_TrackC(ar.trackcs, p) and
    Direction_Correct(ar.trackcs, p))
  Assumptions:
    inv_Area(ar)
    inv_Path(path)
    p in set rng ar.paths
    pid not in set dom (ar.paths)
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
    (forall p in set rng (ar.paths) & ...)
    inv_Path_map(ar.paths ++ {pid |-> path})
```

一方，2 つめの副命題 (2) は以下のようになる．

```
Subgoal(2):
  Conclusions:
    Path_Exists(ar.paths ++ {pid |-> path}, r.paths, r.dr)
    Exists_ATC_for_Route(ar.trackcs, ar.paths ++ {pid |-> path}, r)
    Route_not_Circular(ar.paths ++ {pid |-> path}, r) and
    Path_Connected(ar.paths ++ {pid |-> path}, r.paths, r.dr))
  Assumptions:
    inv_Area(ar)
    inv_Path(path)
    r in set rng ar.routes
    pid not in set dom (ar.paths)
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
    (forall p in set rng (ar.paths) & ...)
    inv_Path_map(ar.paths ++ {pid |-> path})
```

これから副命題 (1) を証明していく．この仮定の中には以下の 2 つの述語がある．

```
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
```

これと似たようなものが結論にあるが，若干結論とは異なっている．実は，結論を証明するには仮定

の inv_Area(ar) を展開する必要がある．仮定の **Step In** を実行することにより，以下の仮定が得られる．

```
  Assumptions:
    (forall p in set rng ar.paths &
         Path_within_TrackC(ar.trackcs, p) and
         Direction_Correct(ar.trackcs, p))
    (forall r in set rng routes & ...)
    inv_Area(ar)
    inv_Path(path)
    p in set rng ar.paths
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
    (forall p in set rng (ar.paths) & ...)
    inv_Path_map(ar.paths ++ {pid |-> path})
```

この中に以下の 2 つの述語が存在していることが分かる．

```
    p in set rng ar.paths
    (forall p in set rng ar.paths &
        Path_within_TrackC(ar.trackcs, p) and
        Direction_Correct(ar.trackcs, p))
```

Path_within_TrackC(ar.trackcs, p) と Direction_Correct(ar.trackcs, p) という結論はこの 2 つの述語を組み合わせることによって得られる．これを行うのが **Fwd. chain** である．これを実行することにより副命題 (1) が証明できた．

副命題 (2) は (1) よりも複雑である．ここでもう一度示す．

```
Subgoal(2):
  Conclusions:
    Path_Exists(ar.paths ++ {pid |-> path}, r.paths, r.dr) and
    Exists_ATC_for_Route(ar.trackcs, ar.paths ++ {pid |-> path},r) and
    Route_not_Circular(ar.paths ++ {pid |-> path}, r) and
    Path_Connected(ar.paths ++ {pid |-> path}, r.paths, r.dr))
  Assumptions:
    inv_Area(ar)
    inv_Path(path)
    r in set rng ar.routes
    pid not in set dom (ar.paths)
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
```

```
Path_Exists : Path_map * seq of Path_id * Direction -> bool
Path_Exists(paths, route, dr) ==
  forall pid in set elems route &
      pid in set dom paths and
      paths(pid).used(dr);
```

図 6.18 Path_Exists の定義

```
  (forall p in set rng (ar.paths) & ...)
  inv_Path_map(ar.paths ++ {pid |-> path})
```

副命題 (2) においても副命題 (1) と同様に **Strip** により分割することができる．この場合 4 つに分かれる．そのうち，Path_Exists に関係する副命題 (2-1) は以下のようになる．

```
Subgoal(2-1):
  Conclusions:
    Path_Exists(ar.paths ++ {pid |-> path}, r.paths, r.dr)
  Assumptions:
    inv_Area(ar)
    inv_Path(path)
    r in set rng ar.routes
    pid not in set dom (ar.paths)
    Path_within_TrackC(ar.trackcs, path)
    Direction_Correct(ar.trackcs, path)
    (forall p in set rng (ar.paths) & ...)
    inv_Path_map(ar.paths ++ {pid |-> path})
```

副命題 (1) と同様に，仮定の inv_Area を **Step In** で展開し，**Fwd. chain** を使うことにより，Path_Exists(ar.paths, r.paths, r.dr) が得られるが，これは目的の結論とは異なっている．そのため，さらに証明作業が必要であり，ここでは仮定および結論の Path_Exists の定義を展開する必要がある．Path_Exists は図 6.18 のように定義されている．

従って，仮定および結論は以下のようになる．

```
  Conclusions:
    (forall pid1 in set elems r.paths &
        pid1 in set dom ar.paths ++ {pid |-> path} and
        (ar.paths ++ {pid |-> path})(pid1).used(dr))
  Assumptions:
    (forall pid1 in set elems r.paths &
        pid1 in set dom ar.paths and
        ar.paths(pid1).used(dr))
```

```
    ...
    inv_Area(ar)
    inv_Path(path)
    r in set rng ar.routes
    pid not in set dom (ar.paths)
    ...
```

この後，まず **Strip** を行い，forall を外して pid1 in set elems r.paths の部分を仮定に持ってくる．これにより仮定の forall の部分と組み合わせて (**Fwd. chain**)

```
        pid1 in set dom ar.paths and ar.paths(pid1).used(dr)
```

を導くことができる．この **Strip** の実行時，2 つの副命題が生成される．1 つめ (2-1-1) は

```
Subgoal(2-1-1):
  Conclusions:
        pid1 in set dom ar.paths ++ {pid |-> path}
```

であるが，これは **Simplify (Expand In)** で

```
        pid1 in set dom ar.paths or pid1 = pid
```

と書き換えられる．ここで or の右辺は仮定にあるから，(2-1-1) が証明されたことになる．
2 つめの副命題 (2-1-2) は

```
Subgoal(2-1-2):
  Conclusions:
    (ar.paths ++ {pid |-> path})(pid1).used(dr)
```

となる．これは pid1 が pid と等しいかどうかで場合分けするとよい．すなわち

```
   (pid1 = pid => path.used(dr)) and (pid1 <> pid => ar.paths(pid1).used(dr))
```

を証明する．このうち and の右側については ar.paths(pid1).used(dr) がすでに仮定にあることから真となる．その結果，

```
  Conclusions:
    (pid1 = pid) => path.used(dr)
  Assumptions:
    pid1 in set elems r.paths
    (forall pid1 in set elems r.paths &
        pid1 in set dom ar.paths and
        ar.paths(pid1).used(dr))
    ...
    inv_Area(ar)
```

```
inv_Path(path)
r in set rng ar.routes
pid not in set dom (ar.paths)
...
```

もう 1 回 **Strip** を使用すると `pid = pid1` が仮定に回るが，これは仮定の最初と最後の行と比べたときに矛盾になる．仮定が偽の場合には結論は真となる[*7]．これで副命題 (2-1-2) の証明がすべて完了し，従って (2-1) の証明も完了したことになる．

副命題 (2) から導かれる他の 3 つの副命題 (2-2∼2-4) も同様にして証明可能である．こうして全ての証明を行うと手数は 44 に上る．この手順を図 6.19 に示す．

この節の冒頭で，実際の証明作業はここで示したものと多少異なると述べた．実際の証明の順序はここに示した通りにしなくても可能である．

自動証明ができず，対話的証明が必要となる証明責務はたいていは複雑となり，ステップも多くなる．仕様が複雑になるに応じて証明も複雑になる．ここで示した例からも分かるように対話的証明の作業には仕様に関する知識と証明の進め方に対する知識が必要である．これがないと，どこで function を展開する必要があるのか，どの証明手順が最適であるかを判断できない．証明未経験のユーザはまずは対話的証明機能について習熟する必要性がある．それでもボタンを押していきながら証明をすることが可能になっているので，HOL でキャラクタベースで直接対話することを考えるとかなり負担は軽減されている．

6.4 考察および評価

6.4.1 実際に見つかった誤り

仕様の誤りがいくつか証明作業中に発見された．これは直接証明機能で発見されたものではない．証明ができない証明責務を調べ，なぜ証明ができないかを調べることにより発見したものである．ここではその例を挙げる．

事後条件の誤り

事後条件において，事後の値を使うべきところ，誤って事前の値を使用してしまった例があった．それが以下の `Del_TrackC` の記述例である．

```
Del_TrackC : Area * TrackC_id -> Area
Del_TrackC(ar, tcid) ==
  mu(ar, trackcs |-> {tcid} <-: ar.trackcs)
pre  tcid in set dom ar.trackcs and
     forall p in set rng ar.paths & p.tc <> tcid
post tcid not in set dom ar.trackcs;
```

[*7] 仮定に追加できるのはすでにある仮定から導きだせるもの，結論の内の一部に限られている．さもなければ，証明として成り立たない．

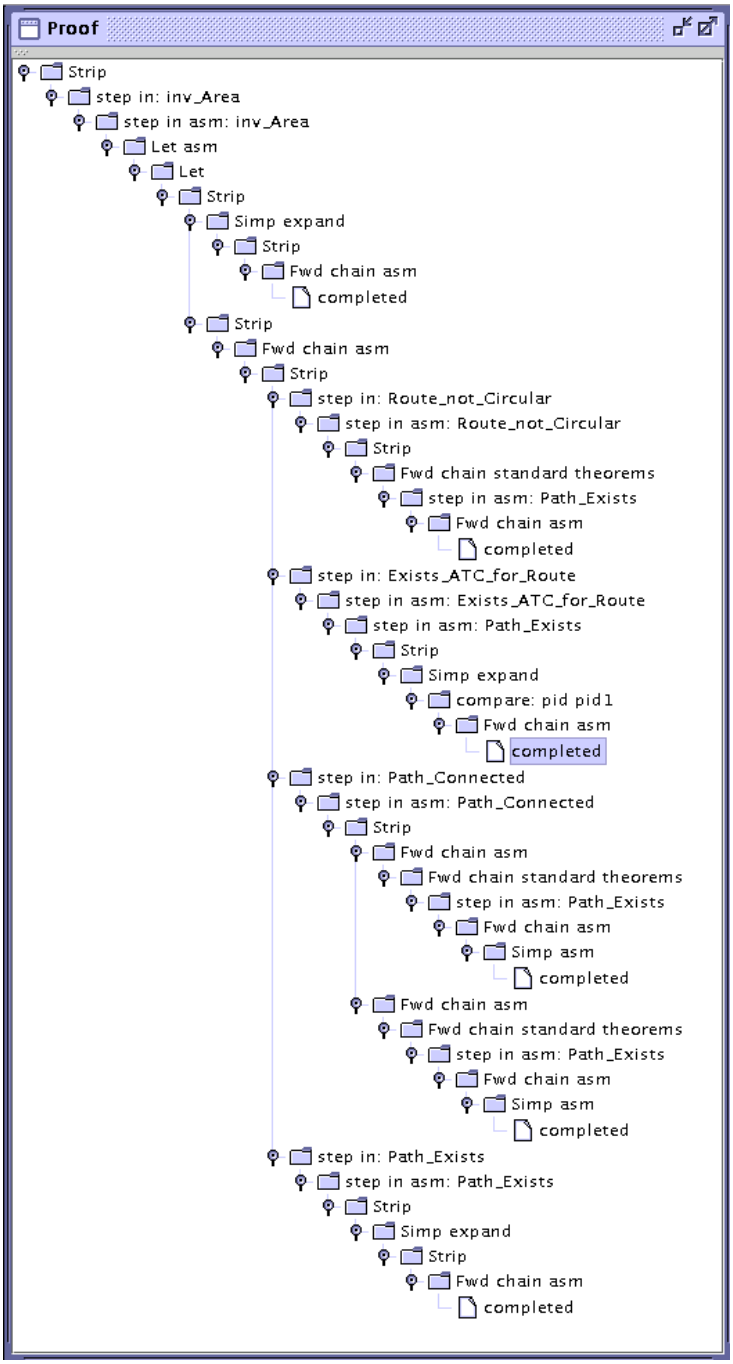


図 6.19　大きな証明木の一例

事前条件と事後条件にある `ar` は同じ値を示している．事前条件では `tcid` が `dom ar.trckcs` に含まれ，事後条件では含まれないのは矛盾していることが分かる．実は事後条件の `ar` は function の出力である `RESULT` を用いるべきところである．この単純な誤りのために証明ができなくなった．

証明機能ではこの誤りを発見してはくれない．しかし，`Del_TrackC` は事後条件に関する証明責務の中で唯一証明に失敗したものであったため，事後条件に問題があることが予想された．そこで事後条件に着目して記述を調べた結果，誤りが見つかったものである．

条件の不足

Add_Route という function は当初以下のように記述されていた.

```
Add_Route : Area * Route_id * Route -> Area
Add_Route (ar, rid, r) ==
  mu(ar, routes |-> ar.routes ++ {rid |-> r})
pre   rid not in set dom ar.routes and
      Path_Exists(ar.paths, r.paths, r.dr) and
      Route_not_Circular(ar.paths, r) and
      Path_Connected(ar.paths, r.paths, r.dr);
```

ここで Area は以下のとおりである.

```
Area :: trackcs : TrackC_map
        paths : Path_map
        routes : Route_map
        ...
inv mk_Area(trackcs, paths, routes, -, -) ==
      ...
    (forall r in set rng routes &
            Path_Exists(paths, r.paths, r.dr) and
            Exists_ATC_for_Route(trackcs, paths, r) and
            Route_not_Circular(paths, r) and
            Path_Connected(paths, r.paths, r.dr));
```

この 2 つを比較すると，Add_Route には Exists_ATC_for_Route(...) が抜けていることがわかる．その結果，Add_Route における不変条件に関する証明責務は証明できないことになる.

この誤りは対話的証明をしているときに見つかった．Exists_ATC_for_Route は対話的証明の際に，結論として現れるのだが，仮定にこの述語がなかったので証明ができなくなったものである.

厳しすぎる不変条件

軌道回路型の不変条件の一部をなす TD_not_Used_for_Insulated_TrackC は，当初は以下のように記述されていた.

```
TD_not_Used_for_Insulated_TrackC : TD * ATBT * set of Joint -> bool
TD_not_Used_for_Insulated_TrackC(td, atbt, joints) ==
  (atbt = <NULL> <=> not td.used) and
  (not td.used <=> forall j in set joints & j.insulated); -- (6.1)
```

ここで (6.1) に問題が見つかった．これは，すべての境界が絶縁されているとき，TD 波が使用されない（有絶縁軌道回路が使用される）という条件である.

以下に示す Del_Joint は軌道回路から 1 つの境界を取り除くものである．

```
Del_Joint : Area * TrackC_id * Joint_id -> Area
Del_Joint(ar, tcid, jid) ==
  let tc = ar.trackcs(tcid) in
  mu(ar, trackcs |-> ar.trackcs ++
                     {tcid |-> mu(tc, joints |-> {jid} <-: tc.joints)})
```

ここで，1 つの境界だけが無絶縁で，その他の境界が有絶縁である軌道回路を考える．この場合，ただ 1 つの無絶縁境界を除くと (6.1) の右辺の値は変わるが，一方で左辺の値は変わらない．これは仕様の矛盾となる．

この矛盾は Del_Joint に事前条件を追加することにより回避することもできるが，この場合は (6.1) の条件が厳しすぎたのが問題であった．そのため，(6.1) を以下のように修正した．

```
(exists j in set joints & not j.insulated) => td.used
```

この矛盾は Del_Joint の対話的証明中に見つかった．TD_not_Used_for_Insulated_TrackC に関連した subgoal が証明されなかったため，この function を調べた結果，この誤りが発見された．

最初の 2 つの例は不注意によって引き起こされた誤りであるが，この例は仕様を記述したときに予想していなかったものである．この意味は重要であり，機械による検査によって，予想していなかった矛盾が見つかる可能性があることを示している．

また，不注意により仕様が誤っていた場合に証明がなされなかったという点も重要な意味を持つ．仕様を記述する側からすると些細なことではあるが，コンピュータの理解は大きく異なる．記述の些細な違いにより仕様の整合性が変わるため，証明は記述が変わるたびにやり直す必要がある．

ただし，実際には証明の手順を保存しておけば，同じ手順で証明し直すことができることが想定される．その場合にはコンピュータに証明を再実行させる時間が問題にはなるが，負担はだいぶ軽減できると考えられる．

6.4.2　証明を容易にするためのヒント

証明を成功させるためのヒントについて考察した．

反例を見つける

今回の証明エンジンでは証明を否定することはできない．このことは成立しない証明責務については常に，成否が判定できないこととなる．その場合，証明責務が否定されるのか，あるいは単に証明が難しいかを見極める必要がある．証明責務を否定するのに最も簡単な方法は反例を見つけることである．それは，多くの場合で全称限定子 forall で量化された形で証明責務が記述されるためであり，反例があれば否定されることとなる．反例の発見は手作業で行う必要があるが，このチェックはインタプリタを使えばすぐに実行できる．

同じ内容は同じ表現で記述する

列を用いるとき，列の index を用いる場合には注意が必要である．2 つの表現を考える．1 つめは

```
forall i in set inds a & f(a(i))        -- (6.2)
```

ここで `inds a` は `a` の index の集合であり，`a(i)` は `a` の i 番目の要素である．もう 1 つの表現は

```
forall x in set elems a & f(x)          -- (6.3)
```

ここで `elems a` は `a` の要素集合である．この 2 つは同じことを言っているが，処理系では違った形で解釈している．1 つめは自然数 `i` に関する述語であり，2 つめは `a` の要素の型を持つ `x` に関する述語である．ここで (6.2) が仮定で，(6.3) が結論と考える．(6.3) は **Strip** により

```
Assumptions:
  x in set elems a
Conclusions:
  f(x)
```

と変形できるから，全体をまとめると

```
Assumptions:
  x in set elems a
  forall i in set inds a & f(a(i))
Conclusions:
  f(x)
```

という Goal が設定されることになる．ここで

```
exists i in set inds a & a(i) = x
```

となる `i` が見つかれば `f(a(i))` の `a(i)` を `x` に書き換えればよいが，実際にはこれは自動で行うのは難しい．

記述の柔軟性は重要であるが，一方で，同じ内容を別の記述とした場合には，一方を他方に書き換える必要があり，自動証明が難しくなる．実用的な観点からすると同じ内容は極力同じ表現にした方がよい．部分的に述語を function として分離するのは可読性という観点だけでなく，証明のしやすさからも重要である．

深すぎる証明木は避ける

自動証明を行う場合，既にある仮定群から新しい仮定を導く場合には，全ての仮定の組み合わせを試し，その結果を仮定群に追加している．そのため，何回かこのプロセスを実行すると仮定が大きくなりすぎる．場合によっては無限ループに陥ることがある．自動証明のプロセスはタイムアウト時間に達すると中止する仕様となっているが，これは自動証明が困難であることも意味している．対話証明では適切な仮定の組み合わせを選択することにより，証明の効率はよくなるが，これを自動で行う

のは困難である．

前節で function を有効に活用すべきと述べたが，一方では function の中身を展開する必要があり，それに伴ってすでにある仮定と組み合わせることになるため，証明が難しくなる傾向がある．記述の見やすさと，証明のしやすさとがここでは相反しているため，証明を自動で行っていくのは困難が伴うと考えられるが，function を活用した場合の証明を簡単にするためには，function の中身を整理し，function 同士の関係を単純化していくよりほかない．

対偶の証明

実行している間に対偶を証明すればよいものが散見された．例えば

$$a_1, a_2, ..., a_n \vdash f(a_1, a_2, ..., a_n)$$

を証明する場合，以下を否定すればよい．

$$\overline{f}(a_1, a_2, ...a_n), a_1, a_2, ..., a_n$$

対偶の証明を行う機能については当初は GUI になかったが，後に追加された．これにより証明が多少は楽になるものと考えられる．ただし，自動で対偶の証明を行うのは難しいと考えられる．

なお，Coq のように排中律が成り立たない場合，対偶の証明を用いることはできない．

6.4.3　評価

証明責務自動生成の有用性

このプロジェクトの開始当初は証明機能はまだ利用できず，自動生成された証明責務を手作業でチェックする必要があったが，それでも反例を見つけ，インタプリタで確かめる方法で仕様の誤りの発見につなげることができた．

ここで，証明責務が自動で生成されるということが重要である．というのも，生成規則にさえ誤りがなければ過不足なく証明責務を生成できるためである．生成規則を誤る可能性を否定はできないが，いったん規則を確立してしまえば誤りはなくなる．この証明責務に関する質は時を経るにつれて確実に向上すると考えられる．

手作業で証明責務を眺めるだけでも，証明の誤りに気づくことがあった．証明責務を手作業で導出するのではなく，自動で生成したからこそ気づくことも多いと考えられる．

自動証明の有用性

自動証明については，まだまだ改善が必要とは考えられる．例えば仮定の組み合わせを自動で試した結果，時間がかかる場面は多かった．また，GUI において定理や補題を導入できないため，対話的であっても証明が完了しない場合が存在する．implicit function の satisfiability の例もそうであるし，今回の仕様では該当しないが数式が現れた場合なども証明が困難である．証明責務で subtype に属するものがあると，証明はほとんどできない状態であり，subtype の問題が出ないように仕様を書き直す必要があった．実際には補題の導入が必要と考えられる．補題の入力は文字入力が一般的に必要で，あくまでもグラフィカルに GUI を構成しようとした場合には対応が難しく，限界があったと考えられる．

なお，タイムアウトにより自動証明が停止する仕様であったために証明が自動で終了しない例もあったが，この検証を実施した当時と現在とでは計算機の能力に違いがあるので，現在このツールが使用できれば性能に多少の違いが出てくる可能性はある．

定理や補題の導入に関しては，バックエンドで動いている HOL に直接アクセスすれば技術的には可能ではあったが，それをユーザに要求するのはハードルが高いといえる．

それでもなお，仕様の品質を上げるという目的であれば使用できるレベルには達していると考えられる．全ての証明責務が自動で証明できるのが理想ではあるが，たとえ全ての証明責務が自動で証明されなくても，証明ができない証明責務に労力を集中できるからである．今回の事例で示したように，証明ができない証明責務から仕様の誤りが見つかる可能性は大きい．

もし，証明ができない証明責務の間で共通の問題が見つかれば，それが証明できない原因の発見につながりうるし，特定の証明責務だけが証明できないとなれば，その部分の固有の問題という可能性が高い．このように自動証明結果を通じて仕様を見直すことにより，証明の矛盾を見つけるという作業はより簡単になりうる．

仕様の理解の必要性

完全な自動証明が使用できれば，仕様に関する知識も証明に関する知識も必要がなく，非常に簡単に扱えるが，特に不変条件に関する証明責務は複雑なことが多く，実際には完全な自動証明は困難な場合が多い．そうなると対話的証明が必要となるが，この対話的証明を成功させるためには，対話的証明機能の使い方の知識だけではなく，仕様そのものの理解が必要になる．これは証明の指示を与える際にどこでどの function を参照したらいいのか，どの型の情報を参照したらよいのか，といったことを念頭に置く必要があるからである．そのため，証明機能が利用できたとしても，仕様をよく理解する必要がある．

そのためには仕様の可読性を上げることが重要である．VDM に関して言えば，仕様記述をプログラムに近い形で記述することが可能であるが，技巧的に記述すればするほど理解しにくくなる．仕様は 1 人で理解できればよいのではなく，他人にも読みやすいように，できるだけ単純にかつ，しっかりとした構造を持たせて記述する必要がある．

例えば，不変条件の述語をいくつかの述語に分割し，それぞれを単純化するといったような工夫が必要である．これは証明を成功させるためにも必要なことである．それは構造が単純になるほど証明も簡単になりうるためである．

6.5 まとめ

信号保安装置のうち，静的な部分への定理証明適用のケーススタディとして，ディジタル ATC 線区データベースの仕様を VDM-SL で記述し，この記述に対して自動証明および対話証明による検証を実施した．90% 程度（188 中 167）の証明責務が完全に自動で証明され，残りは対話的に証明された．一方，仕様の誤りを証明によって見つけることは実際には困難が多いが，それでもいくつかの誤りは自動証明の結果や対話的証明機能で証明を行う中で気付いており，証明機能が完全に自動でできなくても一定の有効性を持つことを確認した．

ここで使用された証明エンジンは研究段階のものであり，例えば仮定の組み合わせを試すことで実

行時間がかかるなどの問題点が見られた．またあくまでも GUI 上で文字を入力することなく対話的証明を行う仕様であったことから，定理や補題を導入することが難しく，必ずしも証明エンジンの機能を完全に活用することはできなかった．そのため，複数の数式を組み合わせて，他の数式を証明するといったことは困難であると考えられ，証明を行う対象によっては必ずしも証明がうまく処理できないことも想定される．

そういう制約はあったが，本章の事例に関しては証明が終了した点では満足のいくものと考える．ただしこの事例においてはじめから 90% が自動で証明されていたわけではなく，仕様の改良および証明エンジンの改良を経てこのような値が得られている．証明エンジンが改良され更に多くの開発に使われれば，もっと使い勝手のよいものができた可能性が高い．

本章の事例を通じて得られた知見をまとめると以下のとおりである．

- 証明責務をながめ，反例を探してインタプリタで確かめることで仕様の誤りを見つけることができた．証明責務の自動生成は仕様の誤りを発見する上で重要であり，自動証明ができなくとも，反例を確かめる手法で仕様の品質向上が可能と考えられる．
- 自動証明ができれば，記述の誤りを見逃さずにすむ可能性がより高まる．これは仕様の品質をさらに向上させる．
- 証明を成功させるためには証明のテクニックを身につけることも必要だが，それ以上に仕様の理解も重要である．そのためには仕様をわかりやすくする努力も必要であり，単純かつしっかりとした構造を構築する必要がある．

VDM での仕様記述は述語の一部を別の function として記述できることから，証明機能が実用化されたならば，かなり強力な仕様検証ツールとなった可能性がある．しかしながら，ここで使用した証明機能は結局研究段階で終わってしまい，一般に使用できるものとはならなかった．そこで次章からの事例においては別のツールを使って信号保安装置への適用を探る試みを行う．

第 7 章

動的な制御部分への定理証明技術の適用 — 単線区間向け自動閉そく装置

前章では VDM で記述した仕様を HOL に変換して VDM 仕様に関する証明責務の証明を行ったが，ここで使用したツールは結局実用化されなかった．VDM がラピッドプロトタイピングには有効であっても，仕様の証明を行うためには VDM とは別の手法が必要となった．

フォーマルメソッドの中でモデル検査法がすでに成果を上げ，注目される存在となっていたが，検証ずみコード生成という観点からすると，方向性が異なっている．そのため，モデル規範型の手法から別の手法を検討した．検討したのは VDM を発展させた RAISE[87] と前述の B であったが，RAISE についても開発が止まっており，サポートが充実しているとは言えなかったため，結局，コード生成機能も謳っていた B を選択した．VDM では概念のみであった段階的詳細化（VDM では Data Reification と呼んでいる）を B では積極的に取り入れていることもその判断を後押しした．現在では Event-B という選択肢も考えられるが，当時はまだ RODIN プロジェクトが始まったばかりの頃で，実用になるツールはなかった．

時系列的にみると実際には第 8 章のブレーキ曲線の検討を先に実施しているが，信号保安装置への適用という観点からすると動的に制御を行う装置への適用可能性を探ることが重要である．

そこで，単線区間向けの閉そく装置という動的な制御を行う装置への定理証明技術の適用の検討について先に述べる．閉そく装置は連動装置に比べると論理規模は小さくなるが，軌道回路による制御，方向回線の鎖錠など，連動装置と似た制御構造を持っており，リレーで構築されていることから，今後連動装置へ展開を考える上で適当な題材と考えられる．なお，検証を実施した最終的な仕様について，その抽象機械の記述を付録 C に示した．

7.1　単線区間向けの閉そく装置の概要

第 2 章で説明した通り，鉄道では列車の衝突や追突を防ぐために，図 7.1 のように線路を一定区間毎に区切り，その区間内に 1 本の列車しか進入させないようにしている．この区間のことを閉そくと呼び，各閉そくに列車を 1 本しか入れないように，列車の在線状況に応じて制御する装置を自動閉そく装置と呼んでいる．さらに単線区間では衝突を防ぐために列車の運転方向を一定とするが，この制御部分も閉そく装置に含まれる．

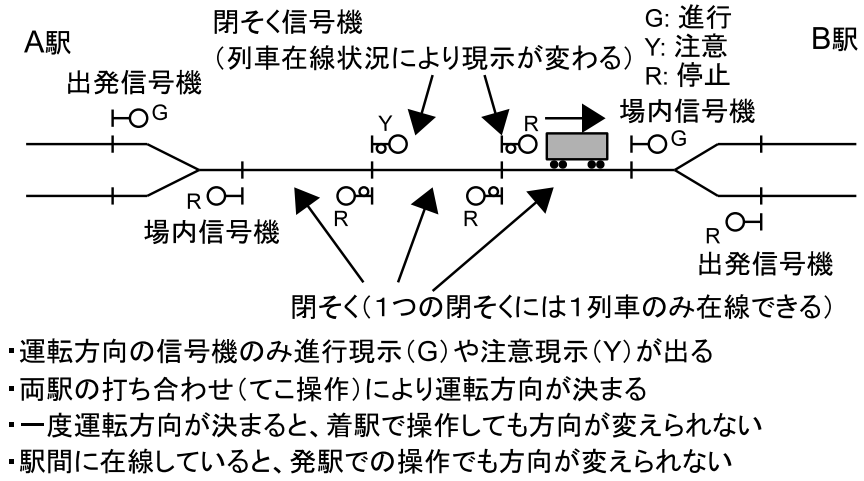


図 7.1　単線自動閉そく装置

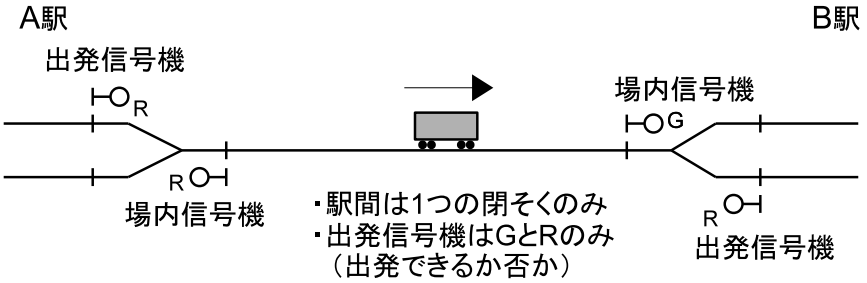


図 7.2　自動閉そく式（特殊）

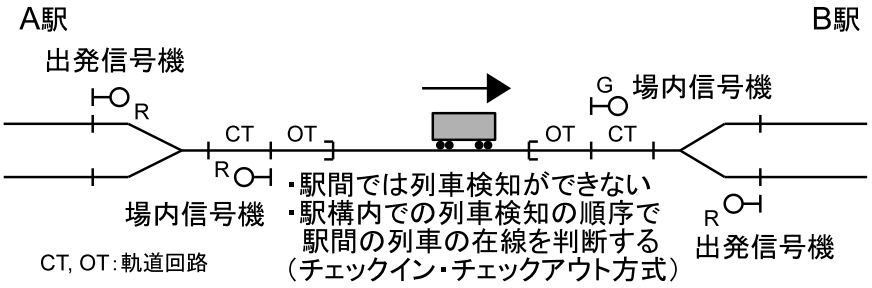


図 7.3　特殊自動閉そく式（軌道回路検知式）

運転方向の制御のうち，要となる方向回線と呼ばれる部分の制御に着目し，共通点が多く見られる単線自動閉そく式（図 7.1），自動閉そく式（特殊）（図 7.2），特殊自動閉そく式（軌道回路検知式）（図 7.3）の各装置を対象に仕様のモデル化を実施した．実際には閉そく装置は単独で存在するのではなく，連動装置の一部として実装されており，継電連動装置の標準結線図 [67, 88] にも閉そく機能に関する記載があるが，ここでは実績のある結線図を検証するのではなく，閉そく機能だけを取り出して装置化した場合に安全性が検証できるかという視点でモデル化を行った．そのためリレーを 1 対 1 で変数に対応させるのではなく，リレーが実現している機能に着目してモデル化した．

閉そく機能の中心となるのは方向回線（駅間に設備され，運転方向の制御を司る 1 対もしくは 2 対

の回線）と運転方向てこ（てことは設定を行うためのレバーのこと）である．上記 3 種の装置において，詳細は異なるが，基本は以下のとおり共通している．

- 両駅の方向てこの向きを一致させることにより方向回線が構成される（取扱上は着駅側から取り扱う）．構成時において方向回線は到着側から電力が送電される．また運転方向リレーと呼ばれるリレーの状態が方向に応じて決定される．
- 一度方向回線が構成されると到着側の方向てこを扱っても運転方向は変わらない．
- 出発側の方向てこは，出発進路（駅から駅間へ列車が進むための進路）の設定状況や，駅間の在線等により鎖錠，すなわちロックされる．この機能は運転方向鎖錠リレーと呼ばれるリレーが落下（無励磁状態となる）することで実現される．この間に方向てこを扱っても運転方向は変わらない．

今回は，両駅が列車を送出する条件にならないこと，着駅でのてこ操作が無効になる等の要件についての検証を行うこととした．ただし，いきなり B を用いてモデル化をするのではなく，日本語の扱いが容易である VDM を併用した．ここで VDM-SL は単一モジュールを前提としており，B に変換することが難しいと判断されたため，本章では VDM++ を用いた．基本的なモデル化および検証の流れは以下の通りである．

- システムを方向回線，起点方駅装置，終点方駅装置の 3 つに分け，その上に 3 つのシステムを統合する全体システムを置く．なお，単線自動閉そく装置についてはさらに軌道回路と呼ばれる部分を追加した．
- これらを VDM++ で記述する．
- VDM++ 記述を B に変換する（手作業にて実施）．
- B のツールで生成される証明責務を証明する．
- B の詳細化を実行する．

そして実際に検証したのは

要件 1 両駅が同時に列車を送出する条件にならない．
要件 2 着駅でのてこ操作が無効となる．
要件 3 列車出発後，到着までの間，運転方向が固定される．

といった項目である．これにより自動閉そく装置の機能が実現され，列車が安全に運行できるかどうかを確認できる．

7.2 自動閉そく式（特殊）のモデル化と検証

7.2.1 自動閉そく式（特殊）の VDM モデル化

まず，図 7.2 に示す自動閉そく式（特殊）の装置についてモデル化を実施した．このシステムでは駅間の閉そくは 1 つに限定される．駅間の列車の在線については有無が分かればよく，位置は問わない．実際にはレールに流れる電流によって列車を検知する軌道回路が設備されている．歴史的に後述する単線自動閉そく式を簡略化したものであるために特殊と名前が付いているが，3 つの装置の中で

表 7.1　閉そく装置のリレー（自動閉そく式（特殊）の場合）

略称	名称	備考
LYR	左方向てこ反応リレー	L 方向発駅に設備
RYR	右方向てこ反応リレー	R 方向発駅に設備
LFR	左方向運転方向てこ反応リレー	R 方向発駅に設置し，RYR と対になる
RFR	右方向運転方向てこ反応リレー	L 方向発駅に設置し，LYR と対になる
FLR*	運転方向鎖錠リレー	FR を保持する
FR*	運転方向リレー	三位（3 つの状態を持つ）
FCR*	運転方向てこ保持リレー	到着側の LFR あるいは RFR を保持
FKR*	運転方向表示リレー	

注: *印は両駅に 1 個ずつ配置

は最も単純な構成となり，運転方向の制御としては最も基本的と言える．

実際の閉そく装置では主要なリレーにはその機能を示す名前がつけられている．表 7.1 に示す [89]．その他，方向回線の制御を行うための方向てこが両駅に 1 つずつ設備される．

おおまかな機能については以下の通りである．LYR, RYR, LFR, RFR は方向てこに関する状態を保持し，方向回線を制御する．FR, FLR は運転方向の状態を保持する．FCR は方向回線の状態により状態が決まり，LFR, RFR を制御する．FKR は方向回線の状態と軌道回路の状態により状態が決まり，FR, FLR を制御する．

リレーの結線は駅ごとに異なるが，標準結線図 [88] を応用することで実際のリレーの結線図が設計されている．今回はこの標準結線図を参考に前節で説明した機能に着目してモデル化することとした．

駅装置のモデルのそれぞれにおいて，運転方向リレーに相当する変数「運転方向」，運転方向鎖錠リレーに相当する変数「運転方向鎖錠」，方向てこに相当する「方向てこ」などを構成要素として記述した．両側の駅装置間では役割が共通であるリレーが多く見られる．そこで VDM++ の記述では駅装置クラスを作成し，その継承として起点方および終点方駅装置クラスを記述した．起点方駅装置における不変条件を図 7.4 に示す．VDM++ において bool 型の true, false といった値を使うこともできるが，日本語の引用型を使うことで意味を明確にできる．図 7.4 の 2 行目は，自分自身の「運転方向」の設定が下り，かつ「反対側方向回線」（反対駅での方向回線設定）が上り，という状態にはならない，つまり要件 1 を示している．次の 3,4 行目では「運転方向」が上り，つまり到着側となっている場合には「進路鎖錠」（自駅から駅間に向かう進路が設定されていると鎖錠となる）が解錠され，「運転方向鎖錠」が解錠されることを示している[*1]．最後の 5〜7 行目は，下りの場合の要件 3 の静的条件を示しており，「運転方向鎖錠」が鎖錠される条件であり，列車が在線しているか，「進路鎖錠」が鎖錠されて列車が出発できる状態であるか，「進路てこ」が設定されている（進路の設定を要求している）場合には，「運転方向鎖錠」が鎖錠されていることを示している．このように日本語が使えることで，可読性が増しており，これは考えを整理する上で重要な点である．

[*1] 少々分かりにくいが，リレーの制御では回路構成上解錠となる．運転方向が上りであれば基本的には運転方向の設定ができないため，問題はない．

```
1  inv
2  not (運転方向 = <下り> and 反対側方向回線 = <上り>) and
3  (運転方向 = <上り> =>
4    (進路鎖錠 = <解錠> and 運転方向鎖錠 = <解錠>)) and
5  (運転方向 = <下り> =>
6    ((方向回線在線 = <在線> or 進路鎖錠 = <鎖錠> or 進路てこ = <設定>) =>
7        運転方向鎖錠 = <鎖錠>))
```

図 7.4 起点方駅装置の VDM++ モデル（不変条件）

```
1  operations
2    public 下り設定 : () ==> 方向回線‘方向型
3    下り設定 () ==
4      (方向てこ := <下り>;
5       if 運転方向 = <上り> and 方向回線在線 = <非在線>
6          and 反対側方向回線 = <下り> then
7         (運転方向 := <下り>;
8          運転方向鎖錠 := (if 進路てこ = <設定> then <鎖錠> else <解錠>));
9       return 運転方向);
```

図 7.5 起点方駅装置の VDM++ モデル（下り方向への設定）

これに，てこの操作や列車の移動に対応する operation や function を加える．VDM++ の operation についても VDM-SL と同じように explicit な定義と implicit な定義の 2 種類の記法があるが，ここでは explicit な定義を行った．図 7.5 に下りに方向てこを設定する行為に相当する operation を例として示す．ここでは「方向てこ」を下りにした上で，「運転方向」が上り，かつ「方向回線在線」（方向回線での列車の在線状態の判定結果を示す）が非在線，かつ「反対側方向回線」が下りの場合に，「運転方向」を下りにし，「運転方向鎖錠」の状態については，「進路てこ」が設定されている場合は鎖錠とし，そうでない場合は解錠としている．なお，図 7.5 では遷移条件を省略したが，基本的には遷移条件は事後条件として記述可能で，`post` というキーワードに続けて記述する．

一方，方向回線は 3 本の回線で構成される 2 対の回線であり（1 本を共通の負極としている），1 対は方向てこを制御するための回線，もう 1 対は軌道回路の条件を照査して運転方向表示リレー（FKR）を動作させるための回線である．これに対応して両駅のてこの状態と軌道回路の状態で方向回線を記述した．

7.2.2 自動閉そく（特殊）の B モデルへの変換

次に VDM++ の記述を B に変換する．まずは変数を変換する．本章の VDM モデルではほとんどレコード型は使っていないが，VDM++ のレコード型は B においては複数の変数として宣言し，別途不変条件を宣言する．また，VDM++ のレコード型に対応して 1 つのモジュールを宣言してしまうのも 1 つの方法である．

そして不変条件も変換する．基本的に VDM の引用型は B の enumerated set に置き換えることができる．図 7.6 に図 7.4 に対応する不変条件を示す．`direction` が図 7.4 の「運転方向」，`oppositedirect` が「反対側方向回線」，`routelock` が「進路鎖錠」，`directlock` が「運転方向鎖

```
INVARIANT
  not(direction = down & oppositedirect = up) &
  (direction = up =>
    (routelock = unlock & directlock = unlock)) &
  (direction = down =>
    (((existence = detect or routelock = lock or routelever = set) =>
      directlock = lock) &
    ((existence = nondetect & routelock = unlock & routelever = unset) =>
      directlock = unlock)))
```

図 7.6　B における不変条件の記述例

```
xx <-- DownSet =
  ANY dir WHERE
    dir : DIRECTION &
    ((direction = up & directlock = unlock) => dir = down) &
    ((direction = down or directlock = lock) => dir = direction) &
    transition(direction, directlock, oppositedirect, dir, directlock, oppositedirect)
  THEN
    directlever := down ||
    direction := dir ||
    xx := dir
  END;
```

図 7.7　B の当初の抽象機械モデルの例（起点方駅装置の operation）

錠」，existence が「方向回線在線」，routelever が「進路てこ」である．

VDM の operation のうち，explicit な定義については，B の抽象機械で逐次代入が使えないため，まずは同時代入できる形式とした上で B の記述に変換する．例えば

```
if a = b then a := c else a := 0;
b := a * 2;
```

のように条件分岐が途中にあるような場合には

```
if a = b then (a := c; b := c * 2)
else (a := 0; b := 0);
```

として，各分岐の中で実行される演算の順序に関係なく同一の結果が得られるようにする．また implicit な定義の場合，B において直接事後条件を表現できないので，図 3.1 の非決定的選択を用い，その条件にある P(X) に事後条件を記述するという手法を取る．

VDM++ の function については，マクロとして定義するか，別途モジュールを定義し，function を適用している箇所については関数型として宣言された抽象変数を使って記述する方法を採る．この抽象変数はこの別モジュールを詳細化することで実装される．

図 7.5 に対応する B の抽象機械における記述例を図 7.7 に示す．ここで非決定的選択を使用し dir という変数を宣言している．dir : DIRECTION は型宣言であり，その次の 2 行は図 7.5 の if 条件が成り立つ場合，dir の値を下りに相当する down とすることを述べている，その次の 2 行の条件は図

```
1  DEFINITIONS
2     transition(dir, d_lo, op, dir2, d_lo2, op2) ==
3    ((dir = down & op = down) =>
4      (((d_lo = lock & d_lo2 = lock) => dir2 = dir) &
5        op2 = op)) &
6    ((dir = up & op = up) => dir2 = dir)
```

図 7.8 状態遷移条件の記述例

7.5 の if 条件の否定を意味しており，if 条件が成立しないときは dir は direction の値と等しいということになる．

図 7.7 の 6 行目に transition という記述があるが，これは事前の状態を記述する変数と事後の状態を記述する変数の間の条件を記述したマクロであり，ここで動的な制約を記述する．マクロということで operation 間で共有可能である．transition の具体的な定義を図 7.8 に示す．図 7.8 の transition の引数のうち，最初の 3 変数は演算適用前の direction，directlock，oppositedierct を想定し，後の 3 変数は適用後の direction，directlock，oppositedierct を想定している．図 7.8 前半の 3〜5 行目は「適用前の運転方向設定が下りかつ反対側の運転方向設定が下りであれば，運転方向が鎖錠されつづけていれば運転方向設定は変わらないし，反対側の運転方向も変わらない.」ということで，これは要件 3 の動的な部分に対応する．最後の 1 行は「適用前の運転方向が上りかつ反対側の運転方向が上りであれば，運転方向設定は変わらない.」ということで，要件 2 に対応する．

図 7.7 の 8〜10 行目には等価代入が並んでいるが，ここで条件を満たす dir を実行後の direction の値としている．最後の代入部分に directlock，oppositedirect がなく，これらの変数は値が変わらないため，transition において事前事後の両方の値として使用している．なお，実際のモデル（付録 C.1 に掲載）においては最後の代入部分を別の抽象機械を include し，その operation を使用しているが，図 7.7 では，説明上それを展開している．

7.2.3 証明責務の生成と証明

証明責務は，Atelier B[32] などの B メソッドの開発ツールを用いれば，仕様から自動で生成される．B の抽象機械において，operation に対する証明責務は，3.2.1 節の（3.1）式で示した通り，I を不変条件，S を一般化代入としたとき，$I \Rightarrow [S]I$ で示される．生成された証明責務の大部分は統合環境からのボタン操作により自動的に証明される．残りは対話証明器を扱って対話的に証明する必要がある． Atelier B の対話証明器については，現在では図 7.9 に示すような GUI が提供されている．証明は基本的には Backward Proof のスタイルを採る．上段中央が結論を表示するパネルで，その下がコマンド入力パネルである．左が証明木の表示で，その下が証明責務の状態を表示するパネル．最下部は仮定の検索結果となる．仮定については全てを表示せず，パターンマッチングにより検索されたものが表示される．前章のツールとは使い勝手は異なり，もともとがテキストベースで作成されていた対話証明器のコマンド入力を支援するものと考えた方がよい．一部のコマンドがボタンで代用できるものの，基本的にはコマンド入力パネルからテキストでコマンドを入力する仕組みである．その代わり，テキストベースで入力できることは問題なく実行できるため，補題の追加や定理の適用，存

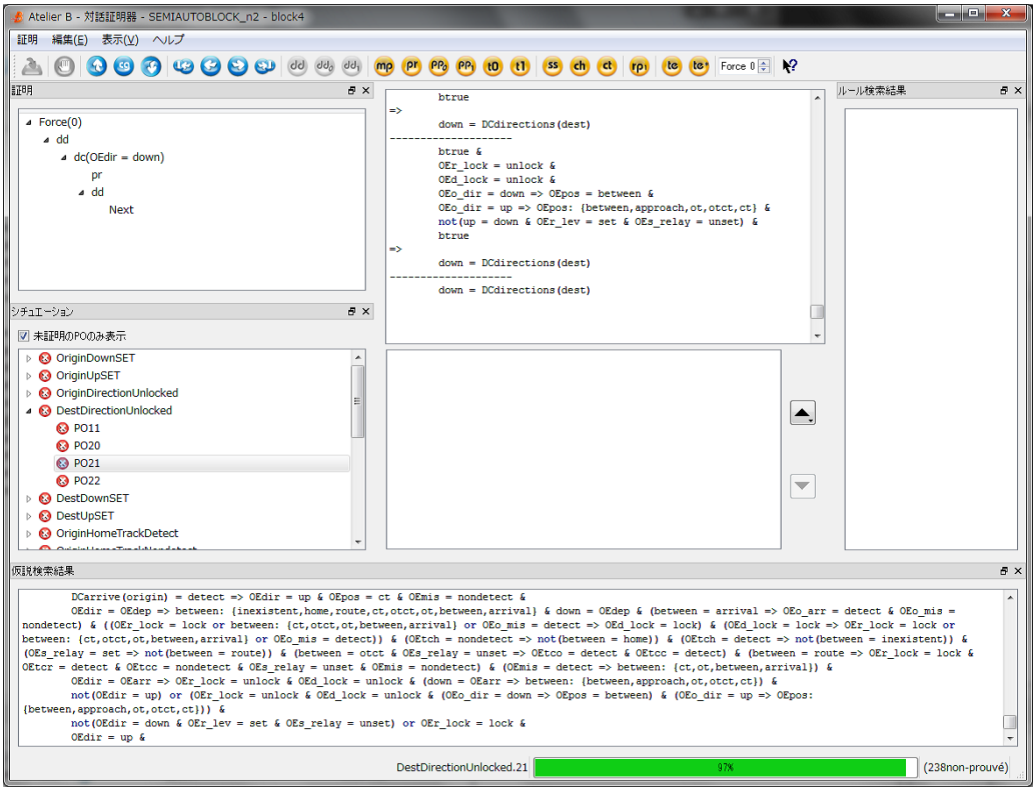


図 7.9　Atelier B の対話的証明 GUI

在限定子に対する証明もできるようになっている．

また，証明状態の管理機能があるため，任意の証明責務を選択し，証明を開始することができるが，同時に 2 つ以上の証明責務の証明を並行して実施したり，複数の subgoal の間を行ったり来たりするようなことはできず，その場合には一旦それまで実行したコマンド列を保存するようになっている．また，証明木については，直接その途中に移動することはできないものの，グラフィカルな操作で証明のコマンド列を部分的にクリップボードにコピーすることができるようになっている．証明器の主なコマンドを表 7.2 に示す．

証明が完了すると，その証明のコマンド列を保存するようになっている．これを Atelier B では User Pass と称している．この User Pass を他の証明責務にも対しても適用を試みることで，同様の証明責務を少ない手順で実行できる．6.4.1 節において，証明の手順を保存すれば負担は軽減できると述べたが，それが実現されている．

前章と同じで記述に不足や矛盾がある場合には証明ができない（不成立とも判定されない）場合がある．このような場合，証明責務が成立するように記述を修正する必要があるが，その度に証明責務は生成し直す．ただし，仕様の修正を行った場合でも一度証明した証明責務と同じ内容であると判断される場合には，その証明はやり直さなくても済むようになっており（ただし証明責務としての ID が変わってしまった場合はその限りではない），User Pass の再適用も可能となっている．

図 7.7 は最終的には図 7.10 のような形となり，不変条件や遷移条件に対応して条件が追加されただけではなく，`directlock` についても値を変更する必要が生じている．

表 7.2　B の主な証明コマンド

コマンド	意味
pr	自動証明を実行.
ba	証明ステップを戻る.
dd	goal が `P => Q` のとき，`P` を仮定に加え，`Q` を新たな Goal とする.
ss	goal を書き換え規則に従って書き換える.
mh	modus ponens.　仮定に `P` および `P=>Q` があるとき `Q` を仮定に加える.
ph	`!x.(x : A & P(x) => Q(x))` および `P(a)` の場合に `Q(a)` を仮定に加える.
dc	与えられる条件式の成立の可否で場合分けをする.
ct	goal の否定を仮定に加え，false を goal とする.
fh	仮定が矛盾している（背反する 2 つの命題がある）ことを示す.
ah	補題の追加.
ar	定理の適用.
se	存在限定子に対して，条件を満たす値を提案する.

```
1  xx <-- DownSet =
2    ANY dir, dir_l WHERE
3      dir : DIRECTION & dir_l :LOCK &
4      ((direction = up & existence = nondetect & oppositedirect = down) =>
5        (dir = down &
6          (routelever = set => dir_l = lock) &
7          (routelever = unset => dir_l = unlock))) &
8      ((direction = down or existence = detect or oppositedirect = up) =>
9        (dir = direction & dir_l = directlock)) &
10     transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
11   THEN
12     directlever := down ||
13     direction := dir ||
14     directlock := dir_l ||
15     xx := dir
16   END;
```

図 7.10　最終的な B の抽象機械の例（図 7.7 の operation に対応)

7.2.4　B モデルの詳細化と検証結果

前節における抽象機械の仕様記述に対して，これを詳細化した演算を記述する．概要で述べた通り，詳細化段階においては operation 実行結果の値域がより狭く，なおかつ実行結果が詳細化前の段階の条件を満たすように記述が求められる．最終的にはプログラミング言語に変換可能な形にまで詳細化し，それを変換してコードが得られる．図 7.7 や図 7.10 では非決定的選択を使用しているが，実装段階では値が決定的に選択される必要があるため，このような記述は行えない．そこで IF 条件を用いて図 7.11 のように記述した．図 7.10 の段階で，ある条件が成立する場合の関係式，成立しない

```
xx <-- DownSet =
  IF direction = up & existence = nondetect & oppositedirect = down THEN
     xx := down;
     IF routelever = set THEN
        directlever := down;
        direction := down;
        directlock := lock
     ELSE
        directlever := down;
        direction := down;
        directlock := unlock
     END
  ELSE
    xx := direction;
    direction := down
  END;
```

図 7.11　B の最終的な実装モデルの例（図 7.10 の operation に対応）

表 7.3　証明責務の数（自動閉そく式（特殊））

module	行数 (B)	証明責務	自動証明	対話的証明	User Pass
Station	209	99	85	14	13
Direction Control	60	21	17	4	4
Whole System	207	243	195	48	24
summary	761	462	382	80	56

場合の関係式という記述の仕方をしている．そのため，これを IF 条件に変更するには，その条件を IF 以下の条件式として，THEN 以降に成立する場合，ELSE 以降に成立しない場合の値の代入を記述すればよい．

最終的には今回の仕様記述に対しては証明責務を全て証明することができた．実際に証明した数を表 7.3 に示す．この数は最初の抽象的な段階での証明から実装段階の証明までを含んだものであり，その大小は仕様の複雑さを示す目安となる．なお，合計については表に記していないモジュールを含むので各行の和とは一致しない．表 7.3 においては最初の段階での B の記述行数についても示した．今回はおよそ 8 割の証明責務が自動的に証明されたことになる．

表 7.3 には User Pass を含めた．実際の証明の実行の手間は User Pass の数に応じて増えるが，この例では全体システムを除くとほぼ対話的証明の数に一致している．モデルの再利用を考えた場合，User Pass を含めて提供を行えばよい．

なお，B においては推論規則（rule）を追加することにより，自動証明できる数が増えるが，rule は検証しない形で追加可能であり，場合によっては証明系の整合性が取れなくなる．そのため，本章での検討では追加を行っていない．

次節以降においては，表 7.3 の証明責務の数を基準に他のシステムを記述した時の証明責務の数がどのように変化するかを，検証していく．

```
1    !ii.(ii : TC => (
2       (ex(ii) = detect => as(ii) = red) &
3       (tr(ii) = tstop => ex(ii) = detect) &
4       ((tr(ii) = t90 & ex(ii) = nondetect) => as(ii) = green) &
5       ((tr(ii) = t45 & ex(ii) = nondetect) => as(ii) = yellow)))
6    !ii.(ii : TC => ((ii + 1 : TC & tr(ii) /= tstop) =>
7       ((tr(ii + 1) = t45 => as(ii) = red) &
8       (as(ii) = red => tr(ii + 1) = t45))))
```

図 7.12 駅間の閉そくのモデル（不変条件の一部）

7.3 単線自動閉そく式のモデル化と検証

7.3.1 単線自動閉そく式とは

自動閉そく式（特殊）に続いて図 7.1 に示す単線自動閉そく式のモデル化および検証を実施した．単線自動閉そく式では，自動閉そく式（特殊）と比べると，駅中間に閉そくが複数あることを想定しており，同一方向であれば異なる閉そくにおいて別の列車を走行させることが可能となっている．軌道回路は閉そく毎に設備される．複数列車の間隔を制御するため，列車が在線している閉そくについては，閉そく入口の信号機を停止現示とし，さらに 1 つ外側の閉そく入口の信号機を注意現示，つまり速度を落とすように指示している．この現示情報を制御するため，列車を検知する軌道回路の送電方向を変えたり（列車に向かって電流を流す），極性を変えたり（前方の閉そくに在線しているときに極性を反転する）することで，レールに情報伝送の役割を担わせている．標準結線図 [67] では表 7.1 のリレーに加えて，UFR（上り運転方向リレー）および DFR（下り運転方向リレー），また下り用の軌道回路送受信器と上り用の軌道回路送受信器が線路沿線の信号器具箱に設備され，方向回線が設定された方向の送受信器が使用される．方向回線は設定された方向の軌道回路のいずれかが列車を検知すると鎖錠される．この現示制御を含めてモデル化を実施することとした．

7.3.2 単線自動閉そく式のモデル化

モデル化においては軌道回路部分について別のモジュールとした．送信の極性を表す `tr`，閉そくの現示を示す `as`, 閉そくにおける列車の在線を示す `ex` を用いた場合，軌道回路における送信制御や現示に関わる不変条件を B では図 7.12 のように記述できる．ここで`!`は全称限定子 $\forall$ の意味である．`TC` は閉そくの番号を示す集合，`t90` および `t45` とは極性が通常か，反転しているかを示す．`tr`，`as`，`ex` は後ろに括弧がついているが，この 3 つの変数は `TC` からそれぞれ極性，現示，在線状態への関数として定義されており，括弧の中はその引数となる．従って図 7.12 の 2 行目は `ii` 番目の在線状態が検知状態であれば，`ii` 番目の現示が赤であるという意味，3 行目は `ii` 番目の軌道回路が送信停止状態であれば，列車は検知状態になるという意味となる．4,5 行目では現示が緑になる条件，6,7 行目では現示が黄になる条件を示している．最後の 3 行は信号現示と隣接する軌道回路の送信との関係を示しており，`ii` 番目が赤の現示であれば，`ii+1` 番目では反転して送信するということになる．そうすると，`ii+1` 番目に列車がいない時は黄色が現示されることになる．

表 7.4　証明責務の数（単線自動閉そく式）

module	行数 (B)	証明責務	自動証明	対話的証明	User Pass
Station	217	51	29	22	14
(refinement)	188	77	77	0	-
(implementation)	297	29	29	0	-
Direct Control	69	15	13	2	2
(refinement)	75	31	27	4	4
(implemenataion)	50	2	2	0	-
Track Circuit	118	21	18	3	1
(refinement)	128	134	71	63	51
(implementation)	134	282	162	120	71
Whole System	333	1716	1586	130	53
(implementation)	211	6914	6208	706	54

方向回線に関するモデルについては，いずれかの閉そくで列車が検知されると，方向回線が鎖錠されるという程度の違いであり，自動閉そく式（特殊）と大きな違いはない．なお，モデルについては後述する改良を加えたものを付録 C.2 に示した．

7.3.3　証明責務の数

このモデルに対する証明責務の数を表 7.4 に示す．これらは全て証明することができたが，軌道回路のモデルが複雑となった結果，軌道回路モデル単独で自動閉そく式（特殊）全体以上の User Pass の数となっている．表 7.4 では段階毎に証明責務の数を示した．この増加に対する考察については，7.5 節において加えていく．

7.4　特殊自動閉そく式のモデル化と検証

最後に特殊自動閉そく式（軌道回路検知式）のモデル化と検証を実施した．ここでは装置の内部状態が複雑になるとどうなるかを示す例となっている．

7.4.1　特殊自動閉そく式の概要とモデル化

特殊自動閉そく式（軌道回路検知式）とは自動閉そく式（特殊）の設備をさらに削減した装置であり，図 7.3 に示されるシステムである[*2]．この装置では駅間に軌道回路が設備されないため，駅間の列車の存在を直接検知できない．代わりに，駅間への列車の出入りを検出する CT と OT[*3]と呼ばれる 2 つの短小軌道回路を両駅に設け，方向鎖錠保持リレーと呼ばれるリレーにより，出発側の CT 進

[*2] 特殊自動閉そく式にはもう 1 つ，電子符号照査式と呼ばれるものがあり，車両に積んだ車載器の識別符号を受信することで閉そくを確保する．この装置の別名が第 1 章で言及した電子閉そく装置である．

[*3] CT は閉電路，OT は開電路の軌道回路である．

表 7.5　特殊自動閉そく装置のリレー

略称	名称	備考
LYR	左方向てこ反応リレー	L 方向発駅に設備
RYR	右方向てこ反応リレー	R 方向発駅に設備
LFR	左方向運転方向てこ反応リレー	RYR と対
RFR	右方向運転方向てこ反応リレー	LYR と対
FLR*	運転方向鎖錠リレー	FR を保持する
FR*	運転方向リレー	三位
FSR*	方向鎖錠保持リレー	列車が駅間にいる間運転方向を鎖錠
DNR*, UNR*	運転方向表示リレー	
DRR	相手駅への到着表示リレー	R 方向発駅に設備，到着時に動作
URR	相手駅への到着表示リレー	L 方向発駅に設備，到着時に動作
PCR*	解錠表示リレー	方向回路の送電を転極して到着を伝える
FXR*	落下不能検知リレー	FSR が落下不能の場合に FLR を鎖錠する
OSR*	到着検知リレー	列車が到着時（到着側 OT および CT に在線）に動作
LBOR*	構外退行解除リレー	駅を出発した列車が退行した場合に FSR を解錠する
LBR*	構内退行解除リレー	同上だが，列車が構内に残っている場合に使用
PCSR*	運転方向てこ保持リレー	
LER*	誤出発検知リレー	誤出発を判断すると運転回路表示回路等を遮断

注: *印は両駅に 1 個ずつ配置

入時から到着側の CT 通過時までの間，方向回線を鎖錠する．すなわちチェックイン・チェックアウトにより鎖錠が行われる．この方向鎖錠保持リレーの挙動は構内の軌道回路の列車検知の順序に大きく依存する．方向回線は 2 本で構成されている．列車の到着時には到着側から方向回線に電力を送出する電源の極性を一時的に反転させて出発側に伝達する仕組みを採用している．さらに列車が退行することを考慮し，列車が停車場内に残った状態から退行後に扱う退行解錠ボタンや停車場外に進出した後に退行するために扱う場内代用てこといったものが設備され，取扱いの手順も決められている．出発信号が出ていないのにも関わらず列車が出発しようとするのを検出する誤出発検知リレーも設備される．駅間に列車が在線していないことを保証するために，駅構内のリレーの数はむしろ増えているのが特徴である．表 7.5 におおよそのリレーを一覧に示す．

モデル化では両駅の OT, CT や駅構内の着発線の列車を検知する軌道回路（ホームトラックと称される）と，進路上の軌道回路を記述した．軌道回路が複数あるので抽象機械では変数 `tc` を宣言し，`home`[*4]，`ot`，`ct`，`route` を適用すると，それぞれの在線状況を取得できるようにした．また，実際には様々なリレーの状態によって表現される列車位置を以下のような集合の値で表現することとした．

```
POSITION = {inexistent, home, route, ct, otct, ot, between, approach, arrival}
```

ここで `otct` は OT と CT に列車がまたがっている状態，`between` は駅間に列車が存在する状態，`approach` は到着駅に列車が接近している状態，`arrival` は列車が到着したことを示している．各列車位置において各軌道回路の列車検知状態が変化した場合に，装置が新たな列車位置をどう認識する

[*4] ホームトラックに対応するが，ホームトラックの「ホーム」はプラットホーム（platform）の略である．

```
CTNondetect =
PRE tc(ct) = detect
THEN
  tc := tc <+ {ct |-> nondetect} ||
  IF dir = dep THEN
    IF r_lev = unset & r_lock = lock & tc(route) = nondetect THEN
      r_lock := unlock ||
      IF s_lev = set THEN s_relay := set END
    ELSIF r_lev = set & r_lock = lock & s_relay = unset &
          (pos = home or pos = inexistent) & tc(route) = nondetect &
          tc(ot) = nondetect & mis = nondetect & o_mis = nondetect
    THEN r_aspect := green END ||
    IF tc(route) = nondetect & o_mis = nondetect THEN
      IF r_lev = unset & r_lock = lock & pos /: {ct, otct, ot, between, arrival}
      THEN d_lock := unlock
      ELSIF s_relay = set & pos = otct & tc(ot) = nondetect & mis = nondetect
      THEN d_lock := unlock END
    END ||
    IF mis = nondetect THEN
      IF s_relay = unset & pos = otct THEN pos := ot
      ELSIF s_relay = unset & pos = ct & tc(ot) = nondetect & tc(route) = nondetect
      THEN pos := between
      ELSIF s_relay = set & pos = otct & tc(ot) = nondetect & tc(route) = nondetect
      THEN IF tc(home) = detect THEN pos := home ELSE pos := inexistent END
      END
    END
  ELSIF pos = ct & mis = nondetect THEN pos := between
  END
END;
```

図 7.13　特殊自動閉そく式における複雑な分岐の例

かを，結線図をもとに網羅的に調べ，状態遷移表にまとめた．この中には装置として想定していない動作も含まれる．この場合，事前条件を用いて排除するか，何らかの列車位置を割り当てて，動作を規定する必要があるが，今回は後者の方針を採り，その割当をもとに仕様記述を行った．

その結果，列車検知状態の変化に対応する operation に関しては，列車位置などの状態によって複雑な条件分岐をすることとなった．図 7.13 に B の記述例を示す．これは CT から列車が抜けて非在線状態になったきの状態遷移に対応したものである．列車位置は pos で保持している．また dir は現在保持している運転方向を示しており，dep であれば出発側，arr であれば到着側となる．また，r_lev は進路てこ，r_lock は進路鎖錠リレー，s_lev は場内代用てこ，s_relay はその反応リレー，mis は誤出発検知の状態，o_mis は対向駅の誤出発検知の状態である．s_relay の状態によっても，状態遷移が変わるため，条件分岐がさらに発生している．

また不変条件についても，運転方向の鎖錠条件が増えるだけでなく，位置に応じた変数の制約が加わるため，記述量が増えている．なお，全体のモデルについては後述する改良を加えたものを付録 C.3 に示した．

表 7.6 証明責務の数（特殊自動閉そく式）

module	行数 (B)	証明責務	自動証明	対話的証明	User Pass
Station	604	4979	4485	494	128
(implementation)	577	6341	5386	955	136
Direct control	103	44	26	18	15
(implementation)	95	68	59	9	6
Whole system	342	17502	17100	402	61
(implemenataion)	255	8677	8644	33	12

表 7.7 駅構内モデルに対する証明責務の数（特殊自動閉そく式・抜粋）

operation	Machine		Implementation	
	証明責務	分岐	証明責務	分岐
HomeTrackDetect	39	2	20	2
HomeTrackNondetect	39	2	20	2
RouteTrakcDetect	85	4	61	5
RouteTrackNondetect	1884	137	3835	245
CTDetect	283	28	606	46
CTNondetect	1317	82	1051	92
OTDetect	106	9	78	11
OTNondetect	210	13	160	13

7.4.2 証明責務の生成状況

証明責務の数は表 7.6 のとおりである．駅装置や全体システムで証明責務の数が非常に多い．これは内部状態による条件分岐が増えたためと考えられる．そこで，表 7.7 に駅構内装置モデルの列車検知に関する operation とその中での条件分岐の数を示した．条件分岐の数に対しておおむね 10〜15 程度の証明責務が生成されている．この傾向は単線自動閉そく式の軌道回路モデルと同様の傾向である．

また，全体システムにおいては，表 7.8 に示す通り，一部の operation において極端に多くの証明責務が生成されている．全体システムの仕様記述には表 7.8 の「分岐」に示されるように条件分岐はほとんどない．しかしここでは呼び出した外部 operation の内部で多くの条件分岐が行われている．これを「内部含む」に示した．このように外部 opeartion を呼び出す場合には，呼び出した外部 operation 内での条件分岐も考慮する必要があることが分かる．

表 7.8　全体モデルに対する証明責務の数（特殊自動閉そく式・抜粋）

operation	Machine			Implementation		
	証明責務	分岐	内部含む	証明責務	分岐	内部含む
OriginRouteTrackNonDetect	102	1	137	3	1	1
OriginCTDetect	2446	3	364	1056	2	46
OriginCTNondetect	4442	2	738	1534	2	92
OriginMisDepartureRelease	1248	3	131	1108	2	13

7.5　証明責務の数に関する考察および評価

7.2〜7.4 節に示した通り，3 種類いずれのモデルにおいても生成された証明責務を全て証明することができた．しかしながら，仕様が複雑になるにつれ，証明責務の数が膨大となっている．連動装置などさらに複雑なシステムを考えた場合，やみくもにモデル化を行っても証明しきれない可能性が高くなる．そこで，証明の負担を減らすためにどうすればよいかを，前節までに記述したモデルを対象に検討を行った．まずは単線自動閉そく式での検討をもとに，どのような仕様記述により証明の負担が減るかを検討し，その知見をもとに，さらに多数の証明責務が生成されていた特殊自動閉そく式の仕様記述の改善を図った．

7.5.1　仕様記述と証明責務の数との関係

B の抽象機械において，operation に対する証明責務は，$I \Rightarrow [S]I$ であることはすでに述べたが，今回使用した証明責務生成器は $I = I_1 \wedge I_2 \wedge \cdots \wedge I_n$ とすると

$$I_1 \wedge I_2 \wedge \cdots \wedge I_n \Rightarrow [S]I_k \ (k = 1, \cdots, n) \tag{7.1}$$

という n 個の証明責務を分割して生成する．

なお，I_k が $P \Rightarrow Q$ のような形をしていた場合，

$$I_1 \wedge I_2 \wedge \cdots \wedge I_n \wedge [S]P \Rightarrow [S]Q \tag{7.2}$$

というように P を左辺に移動できる．ここで $Q = Q_1 \wedge Q_2 \wedge \cdots \wedge Q_m$ と記述できる場合，これらも分割される（ただし，自明と判断された場合は除く）ので，証明責務の数を見積もる場合 (7.1) 式における n の値は，個々の節に含まれる含意の右辺についてもさらに分解した上で決定する．

実際の証明責務を分析すると，今回の証明責務生成器では以下の条件によりさらに分割されていることが分かった．

- S の内部で条件分岐が行われた場合，例えば IF P THEN S_1 ELSE S_2 END という記述がなされると

$$\begin{aligned} I_1 \wedge I_2 \wedge \cdots \wedge I_n \wedge P &\Rightarrow [S_1]I_k \\ I_1 \wedge I_2 \wedge \cdots \wedge I_n \wedge \neg P &\Rightarrow [S_2]I_k \end{aligned} \tag{7.3}$$

表 7.9　条件分岐と証明責務の数（単線自動閉そく式・抜粋）

operation	自明	証明責務	計	分岐	M_i の最大	N_{PO}
OriginDownSet	102	197	299	4	1	392
OriginUpSet	52	53	105	4	0	212
DTrackCDown	345	204	549	8	2	896
DTrackCUp	142	147	289	3	1	280
OriginRouteSet	41	0	41	1	0	56
OriginRouteRelease	52	53	105	2	2	162

※不変条件における結合節の数 (N_1)=14

※ include した機械における不変条件の論理和の数 (N_2)=2

のように，証明責務が 2 つに分割される．$I_1, \cdots, I_n$ の全てで分割されるから，IF を 1 回用いる毎におよそ 2 倍の証明責務が生成される．

- `INCLUDE` 文にて他の抽象機械を使用した場合，include する抽象機械の不変条件が左辺に加わるが，これを $J = P \vee Q$ とすると，

$$\begin{aligned} I_1 \wedge I_2 \wedge \cdots \wedge I_n \wedge P &\Rightarrow [S]I_k \\ I_1 \wedge I_2 \wedge \cdots \wedge I_n \wedge Q &\Rightarrow [S]I_k \end{aligned} \tag{7.4}$$

の 2 つの証明責務が生成される．従って include した抽象機械の不変条件等で論理和が 1 回使われる毎に証明責務が 2 倍となる．

- 非決定的選択（ANY X WHERE P THEN S END）を用いたときに $P = P_1 \vee P_2$ のよう記述された場合，include される抽象機械に論理和を使用した時と同様に，証明責務が分割される．

これらの検討から各 operation（初期化を含む）における証明責務の数は以下のように見積もることができる．

$$N_{PO} = N_1 \cdot 2^{N_2} \cdot \sum_i 2^{M_i} \tag{7.5}$$

N_1　不変条件における結合節の数

N_2　include した機械の不変条件内での論理和の数

M_i　operation 内の条件分岐の枝 i の中での論理和の数

単線自動閉そく式について，この 3 つを考慮して証明責務の数を見積もった．結果を表 7.9 に示す．ここでは枝ごとの M_i を示す代わりに目安として M_i の最大値を示した．また証明責務生成器が自明と判断したものを含めた生成数を同時に示した．一部で数が見積もりより多いが，おおよその生成数を見積もれることが分かった．条件分岐の多い operation において証明責務の生成数が多いことが分かる．

ここで，不変条件における論理和の数が 2 となっていることが，証明責務の数を増やしていること，さらにこの不変条件を一般化代入の条件に使用したことで証明責務の数が増えたことが読み取れる．そこで，論理和の使用を減らすことを考える．具体的には $(A \vee B) \Leftrightarrow (\neg A \Rightarrow B)$ であるから，

表 7.10　論理和の除去による証明責務の数の違い（単線自動閉そく式）

(a) 除去前

module	自明	証明責務	自動証明	対話的証明	User Pass
TrackCircuit	107	21	18	3	1
(refinement)	95	134	71	63	51
Whole System	1901	1716	1586	130	53
(implementation)	N/A	6914	6208	706	54

(b) 除去後

module	自明	証明責務	自動証明	対話的証明	User Pass
TrackCircuit	79	13	10	3	1
(refinement)	89	128	64	64	48
Whole System	374	399	327	72	53
(implementation)	2893	1036	811	225	42

これによって書き換える．その結果を表 7.10 に示す．全体システムの証明責務が 1/4〜1/6 になった．このように証明責務の数が条件分岐の枝の数と不変条件内の論理和に大きく影響されることが分かった．

なお，全体システムの実装部分も大きく減っているが，これは実装部であっても外部モジュールについては抽象機械の表現を使用するためである．従って，外部から利用されるモジュールにおいては，論理和を不変条件に使用する回数を積極的に減らすとよいことが分かる．

7.5.2　User Pass を減らすための記述法の検討

証明責務の数は条件分岐と不変条件内の論理和の数が影響しており，これを少し検討し直すだけでも証明責務が劇的に減る場合があることを示した．ただし，改めて表 7.10 を見ると，証明責務の数の減少と比べて User Pass の数はほとんど変わりない．一方で大量の証明責務が生成されているのにも関わらず，大部分が自明と判定される例もある．

Atelier B の処理はモジュール毎の証明責務が増えると，証明責務の整理に時間を要し，処理能力が低下する．例えば自動証明の処理を一度に実行できるのは 3000 程度であり，証明責務が 10000 を超えると生成処理にも支障をきたしてくる．また，証明責務が増えると，証明しようとする内容の見通しが悪くなるため，証明責務を減らす検討は必要である．しかし実質的な手間は User Pass の数に比例するので，証明の手間を減らすには別の検討が必要である．

証明の手間を減らすためにはどうすべきかを考察する．実際に証明すべき式は，前節で示したとおり，$I \Rightarrow [S]I_k$ であって，不変条件の各項を一般化代入で変換したものである．それが不変条件の一部に一致すれば自明となる．また，$S = (P \Rightarrow Q)$ のような場合は $\neg P$ が I に含まれていればよい．こう考えると User Pass を減らすためには，$[S]I_k$ を不変条件 I の一部に極力一致させればよいことが分かる．このためにはいくつかの方法が考えられる．

方法 1 $[S]I_k$ を I の一部に一致させるために，積極的に変換後の変数が満たす制約として不変条件式の全部あるいは一部を与える．それには非決定的選択を用いればよい．この一般化代入は $\forall X.(P \Rightarrow [S]I)$ を意味するが，ここで P に $[S]I$ を加えれば $[S]I \wedge \cdots \Rightarrow [S]I$ とできる．ただし P を満たす値が存在する必要があり，その範囲で条件を加えないと，詳細化段階で証明不能となることに注意が必要である．基本的には一般化代入で変換する変数が関係する不変条件式を加えればよい．

なお，外部モジュールの変数を用いて不変条件を記述した場合は，外部モジュールの変数の変更を operation を通じて行う必要があるため，非決定的選択を用いた条件の追加は困難である．

方法 2 不変条件が $P \Rightarrow Q$ と記述できる場合は IF～THEN～ELSE 等の条件式に P を加える．そう考えると，基本的には条件分岐を使う場合は条件式を極力不変条件の含意の左辺に合わせる方がよい．

方法 3 表明 (assertion) を与える．

条件を表明として与える方法がある．表明とは不変条件から導出可能な式を証明の補題として宣言するもので，仕様中に ASSERTIONS と宣言して記述する．一般化代入内で補題を与える表明付き代入というものもある．IF 条件や非決定的選択と併せて使用ができ，IF 条件の条件式から補題を作ることも可能である．

ここで，補題を与えることにより証明が成功する可能性の高い書き換え規則例を示す．

```
(tc <+ {route |-> detect})(ct) = tc(ct)
```

これは特殊自動閉そく式にて使用した記述で，列車検知状態を示す写像 `tc` に `route` から `detect` への対応を上書きするものである．`ct : dom(tc)` かつ `ct /: dom({route |-> detect})`（`/:` は左辺が右辺に属さないの意）であれば成り立つ命題であるが，自動ではこの書き換えはなされない．不変条件に `tc(ct)` という記述がある場合に `tc := tc <+ {route |-> detect}` という代入を行うと，この不変条件の `tc` を `tc <+ {route |-> detect}` に書き換えたものが証明責務の一部となるが，この部分も書き換えられない．

そこで，これを表明として与えることにより，書き換えが可能になる．これにより証明される証明責務が多く存在する．表明に記述された命題の証明は別途必要であるが，各証明責務で毎回証明するのではなく，表明を記述した箇所で証明すればよい．その他にも `xx : {ot, ct, ...}` というような集合を使用した関係式や，自然数を使った数式にも表明の使用が有効と考えられる．

今回の実施例について，仕様記述と実際の User Pass の中身を再検討すると，仕様を以下の通りに記述していた．

- 個別モジュールにおいては，非決定的選択で演算を記述し，不変条件そのもの，あるいはその一部を条件に加えていた．これは不変条件を満たすものを明示的に与える意図で行っているが，結果的に方法 1 を実践していたことになる．表 7.4 によれば，個別モジュールでの証明が少ないため，方法 1 を使用すると，少なくとも抽象機械の証明の手間は省けることが分かる．
- 外部モジュールを取り込んで構成した全体モジュールで，証明責務が多い．ここでは operation 呼び出しを用いて演算を記述すると，非決定的選択を使用できないため，その外部 operation

表 7.11　対話的証明の数を減らす改良の適用結果（単線自動閉そく式）

(a) 改良前

module	自明	証明責務	自動証明	対話的証明	User Pass
Station	108	51	29	22	14
TrackCircuit	79	13	10	3	1
Whole System	374	399	327	72	53
(implementation)	2893	1036	811	225	42

(b) 改良後

module	自明	証明責務	自動証明	対話的証明	User Pass
Station	193	31	31	0	-
TrackCircuit	※改良前に同じ（修正なし）				
Whole System	764	340	312	28	20
(implementation)	2893	1212	1085	127	18

内の演算において値が書き換わるかどうかに着目して条件分岐を記述していた．

- 外部モジュールを取り込んだ場合に，対話証明を行った証明責務の多くは，条件分岐における条件と不変条件内の含意の左辺の一致が判定できないために，自動証明できないものが多かった．従ってこれらの一致が判定できれば自動証明できる数を増やすことができる．

このような結果から，仕様記述の改善には前述の方法 1〜3 をさらに進めればよいと判断される．外部 operation が関連する変数については，その外部モジュール内での条件分岐に着目し，条件分岐を記述する．条件分岐と表現が一致させられない場合には，分岐前後に ASSERT 文を挿入することで表現を一致させるようにする．そして， 一通り仕様記述した後は，一旦ツールにより証明責務の生成状況や，自動証明で残った証明責務を確認することにより，足りない条件式をチェックし，追加していく．

単線自動閉そく式について，軌道回路モジュールでは，対話的に証明が必要な命題のみが残っていて，改良の余地が少ないため，駅装置のモジュールを中心に修正を行った．結果を表 7.11 に示す．駅装置モジュールの証明責務は 51 から 31 にまで減少し，対話的証明は 22 から 0，つまり全て自動で証明されるに至った．全体モジュールについても併せて修正を行った，証明責務の数は 399 から 340 に減少し，対話的証明は 72 から 28 にまで減少させることができた．

一方でいずれも自明とされる証明責務の数は増えている．条件分岐を使うことは，場合分けを仕様記述において明示することにより証明を容易にしている面があることを意味しており，条件分岐の使用等による証明責務の増大を気にしすぎる必要がないことを示している．

7.5.3 詳細化（実装）段階での証明責務の数と戦略

抽象機械をいかに証明の手間を省きつつ，詳細化し，実装に変換するかについては，多くの議論がある．特に自動で詳細化を行う技法も研究されており，文献 [35] の事例では自動詳細化を併用していることが紹介されているほか，現在では Atelier B にも自動詳細化を行う BART(B Automatic Refinement Tool) と呼ばれるツールが付属する．このツールは，詳細化のルールとパターンマッチングを行い，それに従って詳細化後のモジュールを自動生成するものである．

抽象機械の表現を詳細化する技法については，既存の議論に譲るが，ここでは，抽象機械で非決定的選択を用いて記述したものをいかに決定的なものとしていくかという点に絞って議論を行う．

詳細化段階の証明の負担を減らすためには，抽象機械とは別の戦略が必要である．最終的には演算を決定的な記述にする必要性から，非決定的選択を用いて条件を仮定に追加するという戦略は採れない．明示的な条件分岐を使用していく必要がある．

ここで詳細化に関する証明責務は 3.2.1 節の（3.2）で示したが，ここで事前条件を抜きに考えると，

$$I \Rightarrow [S_n]\neg[S_{n\text{-}1}]\neg I_n \tag{7.6}$$

と記述することができる．ここで I および I_n は詳細化後のモジュールで宣言される不変条件を示しており，実際には詳細化前の変数と詳細化後の変数を結び付ける関係式である．

ここで S_n で IF 条件を使用すると証明責務が 2 倍となる．また，$S_{n\text{-}1}$ においても IF を使用していた場合，さらに証明責務が 2 倍となる．しかし，条件が一致していれば，抽象機械の時と同じ理屈で自明な証明責務が増えるから，可能な範囲で条件分岐を詳細化前後で一致するようにすることで，証明責務の生成数を抑えることが可能となる．

例として，全体システムに記述されている DTrackCUp という operation 内で使用されている次の文を考える．

```
IF ran(DN.existence <+ {xx |-> nondetect}) = {nondetect} THEN
```

実装段階ではこのままではプログラムに変換できないため，

```
DN.existence := DN.existence <+ {xx |-> nondetect};
IF detect : ran(DN.existence) THEN ex := detect ELSE ex := nondetect END;
IF ex = nondetect THEN ...
```

と置き換えられている（実際には operation 呼び出し等を行っているが，ここではそれを展開している）．その結果，実装段階最後の行の IF 文の条件は以下と等価となる．

```
not(detect : ran(DN.existence <+ {xx |-> nondetect}))
```

しかし，これでは抽象機械の IF 文の条件と等価とは直ちに判定できないので，抽象機械，実装の両方の条件で場合分けされてしまう．そこで実装段階の `IF detect : ran(DN.existence)` に続く部分を

```
IF ran(DN.existence) = {nondetect} THEN ex := nondetect ELSE ex := detect END;
```

表 7.12　対話的証明の数を減らす改良の適用結果（特殊自動閉そく式）

(a) 改良前

module	証明責務	対話的証明	User Pass	size
Station	4979	494	128	58.6kb
(implementation)	6341	955	136	270.5kb
Whole System	17502	402	61	7.4kb
(implementation)	8677	33	12	3.3kb

(b) 改良後

module	証明責務	対話的証明	User Pass	size
Station	1353(27%)	41	40(31%)	9.2kb
(implementation)	3217(51%)	28	23(17%)	5.1kb
Whole System	3759(21%)	30	14(23%)	1.8kb
(implementation)	5270(61%)	21	12(100%)	2.6kb

と書き換える．そうするとこの DTrackCUp の証明責務は 229（その他に自明なもの 428）であったものが，142（自明なもの 168）となった．他にもこれにより証明責務が減ったものがあるため，この 1 文のわずかな変更だけで表 7.4 の証明責務が 1036（自明なもの 2893）から 822（自明なもの 2491）に減った．（これらの数字は表にはまとめていない）

このように詳細化段階においては，条件分岐を一致させることで，証明責務を減らすことが可能である．表現で一致できない場合には前節で説明したように表明を挿入することで意図した条件分岐に合わせる方法が有効である．

7.5.4　特殊自動閉そく式への仕様記述改善法適用結果

特殊自動閉そく式の仕様に対し，単線自動閉そく式をもとに検討した 7.5.2 節の改善法を適用した．駅装置のモジュールにおいては，まず，不変条件を整理し，項毎にどの変数が影響されるかを明らかにした上で，個別の演算の修正を行った．詳細化段階については，条件分岐を極力抽象機械と合わせるとともに，条件分岐毎に表現が異なるものについて表明を加えた．全体モジュールについては，極力分岐を外部呼び出しの内部条件に合わせ，適宜表明を挿入していった．実装段階については大幅な修正は行わず，条件分岐箇所における表明の挿入程度に留めた．

結果を表 7.12 に示す．抽象機械について見てみると駅装置の証明責務は改良前の 27%，全体システムでは 21% となった．対話的証明を行った証明責務は抽象機械において駅装置で 41，全体システムで 14 であるが，ほとんどは表明として与えた補題の証明であり，自動証明するのは困難と考えられる．実装段階については抽象機械での条件分岐の削減に伴って生成数が減っているが，さらに表明を加えることにより証明責務の生成数を減らすことができた．ただし割合としては抽象機械ほどは減少はしていない．

User Pass の数については，抽象機械では駅装置で改良前の 31%，全体システムで 23% となって

いる．User Pass の保存ファイル（テキストのコマンド列）のサイズは 58.6kbyte から 9.2kbyte と 1/6 以下となり，1 証明あたりの入力テキストでは半分となっている．これは単に証明責務の数が減っただけではなく，補題の導入により少ないステップ数で証明ができたことを示している．また実装段階については，駅装置ではもとの 17% であり，保存ファイルのサイズに至っては 270.5kbyte から 5.1kbyte と劇的に減少している．全体システムでは数は変わっていないが，サイズが小さくなっており，ステップ数を減少できたことが明らかとなった．

実際には表明を追加するということは証明の作業の一部を仕様に記述する形で実施しているということになる．しかし，一旦導入してしまえば，再証明する際にも少ない手間で証明できるし，完全に自動証明されればより再利用性が高まるといえる．

7.6 まとめ

信号保安装置のうち，動的な制御部分への定理証明技術の適用のケーススタディとして，単線区間向け閉そく装置に B メソッドを適用した．3 つの閉そく装置仕様を B で記述し，証明により両駅から同時に列車を送出する条件とならないなどといった項目の検証を実施した．この事例の結果，動的な部分についても定理証明が適用可能であることがわかった．

さらに，3 つの仕様に対する証明責務の生成状況を観察し，証明責務の生成規則に立ち戻って考察を加えることにより，どのように仕様記述を行えば証明の負担が軽減するかを検討し，実際に改善を得ることができた．本章で得られた知見をまとめると以下の通りである．

- 不変条件内での論理和の使用，非決定的代入における論理和の使用により証明責務の数が増える．従って生成数が多い場合には，これらの表現を少なくすることで証明責務の生成を減らすことができる．また条件分岐の使用は証明責務の数を多くする傾向にある．
- 不変条件に用いる含意の左辺と条件分岐の条件の表現を一致させることで，証明責務が自動証明される可能性が高くなる．また非決定的選択を用いて一般化代入適用時の条件に不変条件を取り込むことで自動証明される証明責務を増やすことができる．
- 論理式の種類によっては，自動での書き換えができず，一般化代入で変換した不変条件が，不変条件式を満たすかどうかを判定できないが，これには表明が利用できる．これにより，証明の作業を部分集約することが可能で，負担軽減につながる．

この結果はツールに依存する部分も多く，一般性を持つと結論するのにはまだ十分でないが，ここで得られた知見を参考とすることはできると考えている．また，本章での検討対象は，元々がリレー論理で記述できるものであり，列挙型に属する変数が主となっている．これに対し自然数を用いた数式や集合演算を多用した場合などでは，異なる傾向が見られる可能性がある．次章ではリレー論理では収まらない事例について検討を行う．

第 8 章

数値計算部分への定理証明技術の適用 — ATC ブレーキ曲線

前章では単線区間向けの自動閉そく装置という動的な制御を行う部分に B メソッドを適用し，モデル化および証明済みプログラムの生成を行った．しかし，信号保安装置には単に論理演算を行うだけでなく，数値計算を行うものがある．ATC や ATS の車上装置がその代表例である．定理証明の適用範囲を探る上でこのような数値計算にも定理証明が適用できるのか検討が必要である．そこで，数値計算への定理証明技術適用のケーススタディとして，ATC システムにおける許容速度を求めるプログラムについて B メソッドの適用を試みた．なお，検証を実施した最終的な仕様について付録 D に示した．

8.1　ATC ブレーキ曲線の概要

今回の検討の対象であるディジタル ATC システムのブレーキ曲線について紹介する．第 6 章で説明したように，ATC とは列車の在線状況等に応じて決まる許容速度以下に列車の速度を制御する信号保安装置であり，ディジタル ATC とは情報伝送にディジタル変調を用いることで情報量を増やし，制御を高度化するものである．このシステムでは，従来システムで伝送していた許容速度ではなく，進行可能な位置までの距離（あるいは区間の数等）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．

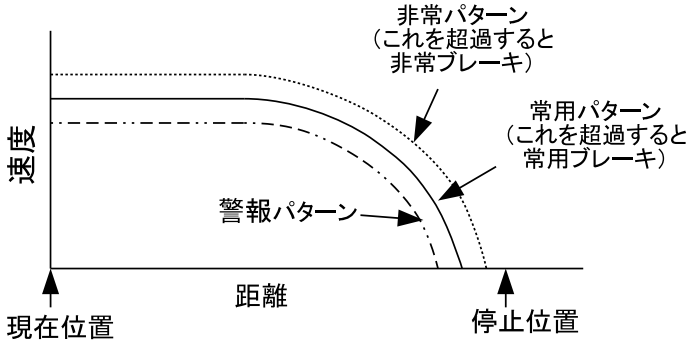


図 8.1　3 種類のブレーキ曲線

ここで図 6.1 に示されるような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．これを算出するプログラムを詳細化によって作成してみる．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．すなわち図 8.1 に示すような 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常に高いが，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非常速度（現在位置における非常パターン速度），常用速度（同様の常用パターン速度），警報速度（同様の警報パターン速度）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ブレーキが完全に効くまでの時間）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 $\leq$ 常用空走時間 $\leq$ 警報空走時間とする．

制約 3 余裕距離は，非常余裕距離 $\leq$ 常用余裕距離 $\leq$ 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．

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．

8.2　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．本節では，最初の関係式の導出部分を示す．

8.2.1　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 8.2 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

```
1  MACHINE
2    BrakeSpeed
3  VARIABLES
4    ta, lo, normalspeed, emergencyspeed, warningspeed, distance, ...
5  INVARIANT
6    ta : NAT & lo : NAT &
7    distance : NAT & normalspeed : SPEED &
8    (ta <= lo => normalspeed  = 0) &
9    (ta > lo => distance <= ta - lo) &
10   emergencyspeed >= normalspeed &
11   normalspeed >= warningspeed ...
12
13 OPERATIONS
14   SetSpeed(tt, ll) =
15   PRE tt : NAT & ll : NAT
16   THEN
17     ta := tt || lo := ll ||
18     IF tt <= ll THEN
19       normalspeed := 0 || distance := 0 ||...
20     ELSE normalspeed, distance, ... :(
21       distance : NAT & normalspeed : SPEED &
22       (tt <= ll => normalspeed  = 0) &
23       (tt > ll => distance <= tt - ll) ...)
24     END
25   END;
26
27   sp <-- Normalspeed =
28     sp := normalspeed;
```

図 8.2 ブレーキ曲線の抽象機械

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（10,11 行目）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

```
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 8.3　ブレーキ曲線モデルの第 1 の詳細化

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 8.3.2 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 $I \Rightarrow [S]I$ は常に真となる．

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．

8.2.2　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 8.3 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．3.2.1 節で述べた通り，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．

$$ta = ta' \wedge lo = lo' \wedge normalspeed = normalspeed' \wedge distance = distance' \tag{8.1}$$
$$ta' \leq lo' + margin \Rightarrow normalspeed' = 0 \tag{8.2}$$
$$ta' > lo' + margin \Rightarrow distance' + margin \leq ta'\text{-}lo' \tag{8.3}$$

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する．

$$ta \leq lo \Rightarrow normalspeed = 0 \tag{8.4}$$
$$ta \leq lo \Rightarrow distance = 0 \tag{8.5}$$
$$ta > lo \Rightarrow distance \leq ta\text{-}lo \tag{8.6}$$

L は (8.2),(8.3) 式かつ

$$ta' \leq lo' + margin' \Rightarrow distance' = 0 \tag{8.7}$$

を満たすように値を代入することを意味する．I は (8.4), (8.6) 式となる．そうすると $[K]\neg J$ は (8.4), (8.5), (8.6) が成り立つ（つまり，詳細化前における代入後の関係式を満たす）全ての値に対して (8.1), (8.2), (8.3)（詳細化前後の関係式）が同時には成立しないことを意味する．この否定は (8.4), (8.5), (8.6) を満たす値の中に (8.1), (8.2), (8.3) を同時に満たす値が存在する，つまり詳細化前における代入後の関係式を満たす値の中に，詳細化前後の関係式を満たす値が存在する，という意味となる．これに L による変換を適用するから，結局この証明責務は，全ての (8.2), (8.3), (8.7) を満たす値，つまり詳細化後における代入後の関係式を満たす値に対して，(8.1)〜(8.6)，すなわち詳細化前後の関係式と詳細化前における代入後の関係式を満たす値が存在することを意味する．言いかえると，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する必要がある，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 (8.4), (8.5), (8.6) は以下の理由で成立する．$margin \geq 0$ により (8.2) が成立すれば (8.4) が，(8.7) が成立すれば (8.5) が成立する．(8.6) に関しては，$ta' > lo' + margin$ の範囲では，(8.3) と $margin \geq 0$ により成立．また $ta' > lo', ta \leq lo$ の範囲では (8.7) により (8.6) 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していることが分かる．

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．

表 8.1　詳細化の内容

module	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 とブレーキ曲線毎の応答時間 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.

8.2.3　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 8.1 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 8.1 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毎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，16bit 整数の値を取るとすれば，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 65km（または 32km）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 8.2 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，1km を超える場合には 10m 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 単位で表されるが，360km/h=100m/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 0.1km/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．例えば速度 $v\times$0.1km/h, 空走時間 $t_i$ms としたときの空走距離（m）は $v/10\times1000/3600\times t_i/1000 = v\cdot t_i/36000$ 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 0.1km/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，その時の制約は

```
((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 8.4 節で説明する．

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 8.4 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 8.5 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 SetSpeed においても normalspeed を使用しない形とした．

```
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 8.4　詳細化 7 における不変条件と operation

8.2.4　実装

最終段階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，前述した通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，

```
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 8.5　詳細化 8 における operation

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一旦終了し，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 8.6 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節で与えるが，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．実装については次節での CalBrakeSpeed の定義の中で実施していく．

8.2.5　証明責務の生成と証明

各詳細化毎に生成された証明責務の数を表 8.2 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概ね 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

```
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 8.6　第 1 段階のブレーキ曲線計算の実装

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 や Distance が実装されて初めて実現可能となる．

8.3　再モデル化（計算アルゴリズムの決定）

前節において関係式が求まったので，次に関係式の実装を行う．ここでは，関係式をプログラムコードのイメージに近い形に変換していく．

8.3.1　抽象機械の記述

抽象機械 CalBrakeSpeed の仕様を図 8.7 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 8.8 に示すが，詳細については 8.4 節で説明する．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

表 8.2　第 1 段階における証明責務の数

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

```
1  MACHINE
2    CalBrakeSpeed
3  VARIABLES
4    normal_speed, ···
5  INCLUDES
6    Distance
7  INVARIANT
8    normal_speed : LOCATION --> (GRAD --> SPEED) &
9    !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
10   (loc0 <= cmargin_n => normal_speed(loc0)(gr) = 0) &
11   (not(loc0 <= cmargin_n) =>
12     normal_speed(loc0)(gr) = max({sp | sp : SPEED &
13      sp * (c_interval + delay_n)/36000 + distance_n(sp, gr) <= loc0 - cmargin_n})) &...
14
15 OPERATIONS
16   sp <-- Normal_Speed(loc0, gr) =
17   PRE loc0 : LOCATION & gr : GRAD
18   THEN
19     sp := normal_speed(loc0)(gr)
20   END;
```

図 8.7　第 2 段階のブレーキ曲線抽象機械モデル

LOCATION * GRAD --> NAT の形式を持つ 2 つの引数を取る関数（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 8.4.1 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．

```
1  MACHINE
2    Distance
3  VARIABLES
4    distance_n, ...
5  INVARIANT
6    distance_n : SPEED * (0..(MaxSpeed * MaxSpeed)) &
7    !gr.(gr : GRAD => distance_n(0, gr) = 0) &
8    !(sp1, sp2, gr).(sp1 : SPEED & sp2 : SPEED & gr : GRAD =>
9        (sp1 >= sp2 => (distance_n(sp1, gr) >= distance_n(sp2, gr)))) &
10   !(sp, gr1, gr2).(sp : SPEED & gr1: GRAD & gr2 : GRAD =>
11       (gr1 >= gr2 => (distance_n(sp, gr1) >= distance_(sp, gr2))))
12
13 OPERATIONS
14   dn <-- NormalDistance(sp, gr) =
15   PRE sp : SPEED & gr : GRAD
16   THEN
17     dn := distance_n(sp, gr)
18   END;
```

図 8.8　停止距離関数を定義する抽象機械

8.3.2　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 8.7 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 8.6 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り返し演算が必要となる．B メソッドにおいては，繰り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，繰り返し演算はこの 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` ではループ実行毎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．この構文に対して，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].

$$
\begin{array}{l}
I \wedge \\
\forall x \cdot (I \wedge P \Rightarrow [S]I) \wedge \\
\forall x \cdot (I \Rightarrow V \in N) \wedge \\
\forall x \cdot (I \wedge P \Rightarrow [n := V][S](V < n))
\end{array}
$$

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 S によって変換し，その後 n をもとの 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り返し回数が増え，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 8.9 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．例えば値の範囲を 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` の値の時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 進表現した時のビット数となる．なお，4 行目において `VAR` で始まっているが，ここでは局所変数の定義を行っている．この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 8.9 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)` 以下の値は常に `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)` で無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．

```
1  sp <-- Normal_Speed(loc0, gr) =
2   IF loc0 <= cmargin_n THEN sp := 0
3   ELSE
4     VAR ll, hh, ii IN
5       ll := 0; hh := MaxSpeed; ii := 0; sp := 0;
6       WHILE ll /= hh DO
7         sp := (ll + 1 + hh) / 2; ii := 0;
8         ii <-- NormalDistance(sp, gr);
9         ii := sp * (c_interval + delay_n) / 36000 + ii - loc0 + cmargin_n;
10        IF ii <= 0 THEN ll := sp
11        ELSE sp := sp - 1; hh := sp END
12      INVARIANT
13        ii : INT & ll : 1..MaxSpeed &
14        hh : 1..MaxSpeed & sp : ll..hh &
15        ll : {sp1 | sp1 : 1..MaxSpeed &
16        sp1 * (c_interval + delay_n)/36000 + distance_n(sp1, gr) <= loc0 - cmargin_n} &
17        (hh + 1) /: {sp1 | sp1 : 1..MaxSpeed &
18        sp1 * (c_interval + delay_n)/36000 + distance_n(sp1, gr) <= loc0 - cmargin_n}
19      VARIANT
20        hh - ll
21      END
22    END
23  END;
```

図 8.9　ブレーキ曲線計算の実装

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 の要素が常に A に含まれ，C の要素が常に B に含まれることを示せばよい．B の要素が常に 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．

8.3.3　証明

CalBrakeSpeed についても証明責務を全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．今回は当てはまらないが，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．

表 8.3 第 2 段階における証明責務の数

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

ここで `CalBrakeSpeed` の詳細化は 3 段となったが，各詳細化毎に生成された証明責務の数を表 8.3 にまとめた．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わかる．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わかった．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，ループ実行毎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 つのループ毎に $6 \times 2 = 12$ 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．例えば `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．このうち例えば `(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．

8.3.4 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，その他の余裕距離等を考慮した上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うと，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得ないと考えられる．

```
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 8.10　停止距離関数の詳細化

8.4　補助関数の実装

8.2，8.3 節において distance_n という関数型の抽象変数と 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．

8.2 節の 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，こちらは関数型で定義した．抽象機械の記述は図 8.8 に示されている．!は全称限定子 ∀ 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．また次の 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，全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．

8.4.1　詳細化における計算

停止補助関数の第 1 の詳細化は，図 8.10 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

```
1  dn <-- NormalDistance(sp, gr) =
2  VAR yy IN
3    IF gr < 0 THEN yy := 0
4    ELSE yy := gr * 72 * Gr1 / Gr2 END;
5    IF yy < N_decl * 72 THEN
6      dn := sp * sp / (N_decl * 72 - yy)
7    ELSE
8      dn := MaxSpeed * MaxSpeed
9    END
10 END;
```

図 8.11 停止距離計算の実装

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を随時判断するのではなく，性能に対して滑走等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 0.1km/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，1km/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，0.1km/h/s 単位で指定される．今回もこれを踏襲する．速度 v×0.1km/h から β×0.1km/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 8.10 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 $9.807\mathrm{m/s^2}$ 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．例えば 5 ‰とした場合 5×353/1000=1715/1000 であるから $0.1\mathrm{m/s^2}$ 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，0.1km/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 0.1km/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

表 8.4　停止距離補助関数に関する証明責務の数

module	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

は図 8.11 のように記述した.

8.4.2　証明責務の数

このモジュールでの証明責務の数は表 8.4 の通りで，while ループを用いていないにも関わらず実装段階での証明責務の数が多い．第 7 章の単線区間の閉そくの例では，場合分けにより証明責務が大量に生成されていたが，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 32bit）の範囲にあることをその都度確認しているためである．例えば計算結果 x が 32bit 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都度この 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 8.11 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 32bit 整数とするため，sp の値を 16bit 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．

他にも除算があると証明が難しくなる．例えば上記（7）の gr * 72 * 353/1000 が 32bit 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 32bit 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．精度をあ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づらく，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．

8.4.3 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伴う証明責務が多く生成されることがわかった．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等を用いれば，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，それに伴い証明責務が多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．

8.5 考察

8.5.1 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 3.0km/h/s，非常 4.0km/h/s とし，余裕距離を警報 200m，常用 100m，非常 50m，

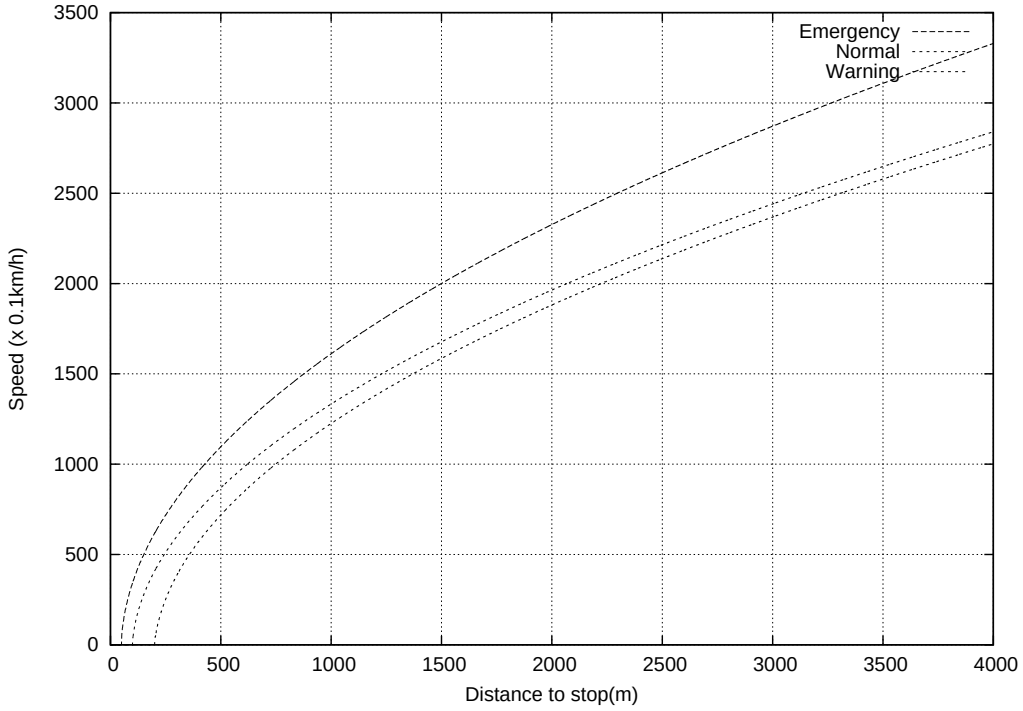


図 8.12　ブレーキ曲線の計算結果例

空走時間を非常 1 秒，常用 2 秒，警報 3 秒とした場合のブレーキ曲線の出力例を図 8.12 に示す．3 つのブレーキ曲線が関係を保たれていること，また放物線状に描かれていることが分かる．

ここで図 8.12 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，値をメモリカード等で設定する場合は，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．

8.5.2　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 32bit の符号付き整数（31bit の正整数）であるため，平方根の値は 15.5bit となるし，3 乗根ならば 10.3bit 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 32bit に収まる必要があり，定義域の方が制約を受ける．正負どちらでも取れるとすると，符号を別として 2 次式の 15.5bit，3 次式の場合は 10.3bit が適用範囲であり，いずれの場合も 3 乗根，4 乗根，あるいは 3 次

式，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，$x = \max(\{y \mid y \in \mathrm{NAT} \wedge y^c \leq a^b\})$ と定義すればよい．ただし，ここでも a^b を 32bit 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 $\log_m n = \min(\{x \mid x \in \mathrm{NAT} \wedge n \leq m^x\})$ 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 $\log_m n = \max(\{x \mid x \in \mathrm{NAT} \wedge m^x \leq n\})$ 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 32bit 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 32bit であるから問題はない．$\log_m x/y$ については $\log_m x/y = \log_m x\text{-}\log_m y$ 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上で提供すれば，より利用しやすくなると考えられる．

8.6 まとめ

論理演算では収まらない数値計算に対する定理証明技術適用のケーススタディとして，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．本章で得られた知見をまとめると以下のとおりである．

- 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，有理数の平方根等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 32bit 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．

第 9 章

まとめと今後の展望

本論文では，信号保安装置に対する定理証明技術の適用事例を検討してきた．静的なデータ，リレー論理による動的な制御部分，リレー論理に収まらない数値計算部分の 3 つの事例について定理証明技術適用のケーススタディを実施した．それぞれを代表する事例について，システムを記述する変数の間に不変条件を有する仕様を記述した．そして，不変条件が維持されることを保証するための証明責務を生成し，自動証明および対話的証明により，それらの証明責務が満たされることを示した．

静的なデータに関する検証が実施できたことにより，システムの制御データなどの検証に定理証明技術が適用可能であると判断できる．また，動的な制御部分の検証が実施できたことにより，信号保安装置の多くの部分を占める，論理回路として設計可能な装置について定理証明技術が適用可能であると判断できる．さらに数値計算部分について検証が実施できたことにより，論理回路ではカバーできない部分についても定理証明技術が適用可能であると判断できる．

これらをまとめると，これまで実施してきた検証技術を応用することにより信号保安装置の多くをカバーできることが分かった．つまり，適用可能性に関して言えば多くの信号保安装置には定理証明技術が適用可能だと言える．この適用可能性を見出したことは本論文における主要な成果である．

また本論文における検証を通じて，定理証明技術および形式手法適用に関する多くの知見を得た．本章では，これまでの検証を通じて得られた知見を概観し，それを踏まえて定理証明技術の鉄道信号分野への応用についての見解を述べるとともに，今後の展開について議論する．

9.1　VDM 記述による定理証明技術の適用の評価

ATC システムのデータベース構造の検証では，バックエンドに the HOL system への変換を行っているものの，VDM 記述の枠組みで検証を実施できた．

第 6 章のデータベース構造の検証を通じて得た知見は以下の通りであった．

- 証明責務をながめ，反例を探してインタプリタで確かめることで仕様の誤りを見つけることができた．証明責務の自動生成は仕様の誤りを発見する上で重要であり，自動証明ができなくとも，反例を確かめる手法で仕様の品質向上が可能と考えられる．
- 自動証明ができれば，記述の誤りを見逃さずにすむ可能性がより高まる．これは仕様の品質をさらに向上させる．
- 証明を成功させるためには証明のテクニックを身につけることも必要だが，それ以上に仕様の

理解も重要である．そのためには仕様をわかりやすくする努力も必要であり，単純かつしっかりとした構造を構築する必要がある．

この知見は B メソッドを使う場合でも有効であると考える．

VDM による記述では，B を用いた場合と比べると，不変条件や function（状態変数を用いた場合には operation となる）適用の事前事後条件を，述語を表す function を利用することで，分割して記述することが可能であるという特徴があり，これにより可読性を向上させることができた．B でこれと同じことをする場合にはマクロを利用するか，別のモジュールを記述する必要が生じるが，述語を示す function と比べると使用しにくい．VDM のこの特徴は，本論文で実施した線区データベースの検証のように要素間の定義関係が複雑になるような仕様では有利と考えられる．

また VDM での記述では，型ごとにそのフィールド間の不変条件を記述することで，不変条件の一部を型の定義に局在化することができる．このことで，不変条件そのものをより小さく記述することを可能としている．

第 7 章の単線自動閉そく装置の検証についても，証明こそ B の枠組で実施したが，仕様検討においては VDM を使用した．これは，述語を function として記述するスタイルが使用しやすいことも理由の 1 つである．この場合も，要素間の連鎖関係が不変条件として書かれることから，仮に VDM での証明支援環境が使用可能であった場合には有効に活用できたものと考えられる．

このように第 6 章および第 7 章の事例を考えると，鉄道の制御においてリレーで構築できる部分については，基本的には VDM を用いてモデル化やある程度の検証は可能であると考えられる．たとえ連動装置のような複雑な場合であっても，第 5 章の記述を拡張していくことにより，連動装置そのもののモデル化は可能であると考えられる．

一方，第 8 章のように，数式を多用し，値の大小の比較結果が後の処理に影響するような処理の仕様においては，証明責務の証明において複数の等式や不等式から別の式を導く必要が想定される．B での証明ではこのような数式の処理については完全に自動とすることが困難であったことから，自動証明に対する補助的な機能として作成された VDM の証明支援環境の GUI では，証明責務の完全な証明は困難であったことが想像される．

また，繰り返しになるが，現在は VDM 仕様を直接証明する手段がない．そのため，証明を行うためには B など別の手法で実施したり，第 6 章で説明したような証明支援系への変換が必要である．

しかしながら，そういった問題点を差し置いても，VDM では記述が比較的容易であることから，仕様の整理の上で有利であり，仕様の証明が直接できなくとも，活用はできると考えられる．実際の検証は別の手法によればよい．

ただし，システムの規模が大きくなった場合に，どの程度の検証ができるかどうかについては未知数である．operation に関して明示的なアルゴリズムが示されるのであれば，その結果が定義に関する不変条件を満たすことの検証は可能と考えられる．また，定められた制御データに従って装置が動作するということも示せるかもしれない．しかし，「列車が衝突しない」というような本当に安全であることの条件をいかに明示できるかについては依然として問題として残る．この安全性を考える上では，特殊自動閉そくの検証で部分的に実施したことであるが，仕様の記述に列車の動きを含める必要がある．ここで列車の動きを完全に形式的に記述するには困難を伴う．モデルの健全性は列車やその検知デバイスが想定された動作を行う前提に立っている．列車が想定されない動きをした場合に本

当に安全性を保てるかは保証できない．そのため，より多くの事例を想定した上でモデル化を実施する必要がある．

また，VDM からのコード生成は C++ や Java に限られる．C 言語への変換については手作業が必要である．従って，仕様を忠実に満たす C 言語プログラムが必要であれば，第 7 章のように B などの手法で実施する必要がある．

9.2 B メソッドによる定理証明技術の適用の評価

B メソッドを用いた記述では，仕様を忠実に満たすコードを生成できた．証明支援環境についても，あくまでも GUI 上で行うことにこだわった VDM の例と比べると（GUI によらず直接 the HOL system にアクセスすればテキストベースの対話証明ができるが，あくまでも証明は the HOL system 上で行うため，the HOL system の文法，tactic，さらには VDM から変換される HOL のモデルについての知識が必要であり，使用する上でのハードルは非常に高い），テキストベースの B のほうが対応できる範囲が大きく，第 8 章のような数値計算の事例でも適用可能であった．

VDM-SL もしくは VDM++ にあって，B にない主要な構成要素はレコード型である．これは変数あるいは定数の組に展開し，型の invariant をモジュールの不変条件に展開すればよいから，VDM での記述を B に変換することは，可読性を別とすれば一応は可能である．実用的な観点からすると，第 7 章で述べたように VDM のレコード型を 1 つのモジュールとして記述する方が，VDM と B との間の対応はつけやすいが，いずれにせよリレーで構築できるシステムは B で記述可能であり，前述したような本当の安全が検証できるかという問題は残るものの，一通りの対応は可能であると考えられる．また，ブレーキ曲線の計算のような数値計算も適用可能であったことから，おおよその信号保安装置に関しては B メソッドが適用可能であると考えられる．B メソッドの鉄道への適用に関しては，海外においてはパリ地下鉄 [34] などの事例がすでにある．しかし，例えば特殊自動閉そくのような日本独自の事例であっても適用可能であることを示すことができた．この事例では列車検知が連続的でなく，1 つの列車検知デバイスの動作による状態遷移が列車位置に関する内部状態によって複雑な影響を受ける所が特徴的であり，海外の事例と異なる部分である．そのような複雑な事例であっても B メソッドの適用が可能であり，さらに適用にあたっての知見を得たことには一定の意義があると考える．

ただし，B メソッドを用いた仕様記述は VDM と比べると困難がつきまとう．VDM-SL と比べると B メソッドは大規模システムへの対応が可能と謳われているが，実際には記述にかなりの困難が伴う．例えば大規模化する場合であっても，システムの構造は基本的にはツリー構造とする必要がある．ここである抽象機械を他の複数の抽象機械で include しようとすると，上位モジュールから見て 2 重に include することになるために認められない場合があり，モジュールの再利用性が意外と低い．従って，いきなり B を大規模システムの開発に適用するのは相当の困難を要する．

実際の証明作業に伴う問題，つまり証明責務が大きく増えてしまう問題に関しては，その理由を第 7 章および第 8 章で明らかにした．すなわち

- 不変条件内や非決定的での論理和の使用，条件分岐等を用いることにより証明責務の生成数が増大する．
- 同じことを別の表現で記述してしまうと自動証明できない証明責務が増大する．

- while ループを使用すると，そのループ不変条件に証明責務が増大する．
- 乗除算を用いると，中間値を含めて実装可能な範囲であることを確認するための証明責務が増大する．

このことを意識して仕様を記述することにより，ある程度の証明責務の増大については対応できる．例えば，論理和や条件分岐の除去や表明の使用により証明責務の数を減らしたり，自動証明率を上げたりすることを示した．

しかし根本的には，文献 [35] で示されているように仕様記述を分割し，モジュールに含まれる不変条件を極力小さくし，モジュール間の関係を単純化していく必要がある．また上記で示した複数の抽象機械での include リンクの使用にまつわる問題も，詳細化段階で include をすれば解決する．抽象段階では include を使わず，抽象変数を使って記述を行う．詳細化段階で include を行い，抽象変数と include するモジュールの変数を不変条件で結ぶ．こうすれば，当該の抽象機械を include しようとするモジュールからは include をする部分が隠蔽されるので，問題が解決する．これらの記述法は B-Book には書かれていないテクニックである．

このように B メソッドの適用については，VDM 以上に記述上のテクニックの蓄積，証明支援環境への習熟が必要である．この状況では，なかなか普及が難しい．B による検証そのものが難しいことを考えると，B メソッドを用いる場合に，この手法を広く信号保安装置に適用するためには，ある程度の仕様記述の標準化を行い，証明の User Pass を含めた仕様記述の参照モデルを構築していく必要があろう．その場合，モデルの理解をしやすくするため，参照モデルに VDM 記述を含めることも考えていく必要がある．

9.3　制御データの正当性の検証手法

信号保安装置においては，第 2 章で説明したように制御論理を実行する部分と制御データが分離された構造を採る場合が多い．これは処理プログラムには実績のあるものを使用し，変更部分は駅毎の固有データに限定することで，信頼性向上を図るのが目的であり，これまでも実績を有しているが，一方で，データ（連動装置で言うと連動図表が相当する）が正しいかどうかが問題となる．考え方としては制御データが一定の制約を満たせばよいとするか，制御データと処理プログラムを合わせて検証してよいとするかのいずれかとなるが，後者の場合はデータを仕様に含めて検証する必要があり，データ毎に証明をし直すことになる．B メソッドを用いた場合には，後者ではデータを分離するという形にはならない．それでは前者の立場を取った場合にどうすればよいかを考える．

B メソッドの場合，制御データを定数とする前提とすると，外部から定数データを読み込もうとした場合，その不変条件をチェックする構造が採れないという問題がある．これはあくまでも B の定数は，その定数が定義されているモジュールが実装段階において初期化される際に値が決まるため，ファイルから読み込んで代入するということが許されないためである．

そこで，ROM やファイルから制御データを読み込み可能とするためには，データに相当する部分を B でいうところの Base module とする必要がある．これは定数の制約条件を抽象機械の `PROPERTIES`，変数の制約条件を `INVARIANTS` に記述し，その実装を手作業でコーディングするものである．ここで実装部分を外部の ROM 等からの読み込みとしてコードを記述すれば実装が可能となる．あとは，実装部分で読み込むデータが定数の制約を満たすかどうかをどう検証するかが問題と

表 9.1　モデル検査法による検証例（自動閉そく（特殊））

検証項目	状態数	実行時間
常に列車が送出条件にならない	1265	0.015s
発駅から運転方向を変更できない	1265	0.015s

なる．

この場合の解決策の 1 つとしては制御データを B の実装，あるいは include/import される抽象機械に変換し，定数の制約を満たすことを証明する方法が考えられる．この変換は手作業で実施するのも 1 つの方法であるが，同じ装置のデータということを考えると，データを B に変換するツールを製作したほうがよい．

こう考えると，連動装置などで処理プログラムと制御データが分離される構造を考える場合は，統合的な検証ツールが必要になると考える．ツールにて制御データを設計し，それを B の表現に変換して証明等により正当性を確認し，確認ののち外部ファイルや ROM に変換すればよい．

なお，システムの安全性を保証するために制御データに求められる制約をいろいろ盛り込んでいくと，制約が大きくなりすぎて，証明責務が大量に発生し，定数がその制約を満たすことの証明がしきれない可能性が出てくる．この点に関して，文献 [90] で実際にこのような事態が発生していることが報告されている．この論文では，その問題に関して，ProB[33] と呼ばれるツールを用いることで解決できたとしている．このツールは B で書かれた仕様に関するアニメーションを実行するものであるが，同時に不変条件やデッドロックをチェックすることが可能とされている．

ただし，ProB は実装段階をサポートしていないので，抽象機械の中にデータの性質を記述する必要がある．前述したように，ProB を用いて外部データを検証する際は，外部データを B の抽象機械に変換した上で，検証を実施する．そして ProB にアニメーション[*1]による検査を実行させる．ただし，ProB の検査機能は状態を列挙して検査を実行するものであり，列挙された状態がメモリに収まらない場合には検査しきれない．したがって，記述した表現が状態を列挙しきれなくならないように，構造表現に対して注意を払う必要がある．

制御データが分離されるシステムに関しては，以上で示した方法論を採ることになるが，結局のところ，最大の問題は実装データそのものに依存しない条件をいかに記述できるかという問題に帰着される．装置によっては困難な作業であることが想定されるため，装置ごとに条件の抽出を検討していく必要がある．

9.4　定理証明と自動検査手法との比較

定理証明とは別の仕様検証手法に，自動検査手法であるモデル検査法がある．自動閉そく（特殊）に関して，モデル検査法で検証を実施した事例を表 9.1 に示す．

この例では SPIN を用いて，モデル化を行い，不変条件の保持の確認を行った．

[*1] Model Checking と ProB では称しているが，一般的なモデル検査法とは別の意味であり，このアニメーションという用語の方が本論文の文脈では適切である．

表 9.2　SMT ソルバによる不変条件検証例

モデル	変数の数	検証した演算	実行時間
自動閉そく（特殊）	24	10	0.060〜0.076s
単線自動閉そく	24	24	0.060〜0.076s
特殊自動閉そく	66	34	0.076〜0.092s

この結果を見てもわかる通り，対象によってはモデル検査法であっても正当性の検証が可能であることが分かる．閉そく装置は元々がリレー論理で構成されている上に変数の数が少ないため，モデル検査法で検査可能な範囲に十分に収まる．定理証明によった場合，同様のことをするためには数時間を要することもあり，不変条件の成立の判断だけであれば，モデル検査法の方が格段に速い．

また，SMT ソルバを用いて不変条件の検証を実施した事例を表 9.2 に示す [91]．不変条件の検査を行うためには，演算実行後に不変条件を満たさない例がないということを示す必要があり，非決定的な演算を行う場合は，SMT ソルバで判定できない場合があるが，それを回避するため，決定的なアルゴリズムで記述した仕様を検証している．そのような制約があるものの，検証そのものは高速で終了している．

これらの結果を見る限り，対象によってはモデル検査法を活用するのも 1 つの方法と考えられる．最終的にはコードの正当性を確保するため，B メソッドを使うことが必要であっても，その前段階で，仕様が不変条件を満たすかどうかの確認にモデル検査を補助的に使うことで，仕様の修正を早い段階で行い，実際の証明作業の場面で，証明を進めた段階で誤りを発見し，修正を行って証明を繰り返し実施する手戻りを少なくできることは十分に考えられる．

ただし，第 7 章の閉そく装置のような bool 値や列挙型の変数で記述される対象であればモデル検査法は有効であるものの，第 8 章のような数値計算の事例に関してはモデル検査法は全くなじまない．そこまで極端な例ではなくとも，自然数をカウンタやインデックスとしてではなく，数式の一部として扱う場合にはモデル検査法では途端に処理が難しくなる．

また，過去の連動装置に対するモデル検査法の適用事例では単純なモデル化では進路数が 10 程度であっても検査が終了しないことが報告されている [66]．少し規模が大きくなっても進路数が 100 を超える例は多くあり，さらにこういった駅では軌道回路やポイントも数十に達することを考えると，モデル検査法の単純な適用は難しい状況である．現在の計算環境であればメモリ容量が大きくなっていることからもう少し計算が可能であり，また SPIN を使用するよりは，NuSMV で記号モデル検査を使用すれば改善が可能かもしれないが，それでもモデル化によっては状態爆発が報告されている [63]．

モデル検査法では，状態爆発等で検査しきれない場合はモデル検査が可能なレベルにモデルを抽象化して，その範囲で検証を実施することとされている．またモデルの抽象化技法自体に研究要素があり，そういった抽象化を行うところがモデル検査の面白いところであるとする議論もあるが，こうした場合の正当性は抽象化したモデルに対してなされるのであり，もとの仕様が同様の正当性を持っているかどうかについての検証は厳密にはできないこととなる．また，モデルを分割する手法もあり，連動装置を対象とした例でも文献 [63] などでは，対象を分割する手法が提案されている．ただしこの

事例も小規模な事例であり，さらに大規模な検討事例は国内ではない．海外の事例 [79, 92] については，有界モデル検査での検証となっており，モデルの分割による対処とは多少異なっている．

一方，モデル検査法と定理証明の関係は完全に排他的な関係ではない．モデル検査技法を定理証明に組み込むという手法も数多く提案されている．B の枠組みでも提案がなされており [93, 94]，これにより証明器の性能向上を図ろうとしている．このような取り組みがうまくいくのであれば定理証明の枠組みであっても，モデル検査技法の進歩を取り入れることが可能であり，今後の発展の 1 方向と考えられる．

なお，モデル検査法における記述は状態遷移に着目したものであり，実際への実装コードにはならない．特に状態遷移が非決定的に書かれた場合には，実装段階でアルゴリズムが決定的にならなければならない．そのため B メソッドのようなコード生成に重きを置いた手法はモデル検査法が導入されたとしても依然有効である．

9.5　今後の鉄道分野への定理証明手法の適用について

今後，これまで検討してきた定理証明技法がどの程度信号保安装置に導入されるか考察する．

本論文で取り上げた例を含め，フォーマルメソッドの適用研究として連動装置の検証がしばしば出てくるが，連動装置そのものへのフォーマルメソッドの適用がすぐ行われるかを考えると，実際には困難と考えられる．それは処理プログラムに関しては実績のあるものを使用しつづけるというのが，信号保安装置の安全の考え方であるためである．

第 2 章で説明したように，連動処理の大部分はテーブル形式や結線に対応する論理式表現で示された制御データを処理している状態である．そのため，B メソッドにより作成したプログラムを使う場合は，プログラムを全面的に作り直す必要が生じる．連動処理プログラムそのもの不具合は少ないことから，少なくとも連動装置の制御が既存の連動図表を踏襲する限り，そこをあえてフォーマルメソッドを導入して作り直すという動機は残念ながら乏しいと言わざるを得ない．

逆に連動図表や制御データの検証などにはフォーマルメソッドの適用余地があるように考えられる．実際の連動装置の動作の要となるのは連動図表データであるから，その検証プロセスの信頼性を向上するとなれば，それなりに導入される理由となる．ただし，この場合には検証の対象となるのが個別のデータとなるから，必ずしも定理証明が有効であるとは限らず，モデル検査法の方が入りやすいように考えられる．ここで検証するのはデータであり，プログラムそのものではないため，モデル検査法のようにコード出力機能はなくても問題はないからである．もっともモデル検査法であっても状態爆発を考えると限界があり，結局定理証明の方が有効かもしれない．いずれにせよ，フォーマルメソッドをそのまま使わせるのではなく，ツールを作成しそこでデータを編集し，バックエンドとして，モデル検査法や定理証明用のモデルに変換して検証するという流れがあれば，実用的なものができると考えられる．

連動装置以外の装置，例えば ATC 装置などについても，すでにあるシステムに関しては，既存の装置の置き換えというアプローチを採ると導入のための動機づけに乏しい．B のような手法で信号保安装置にフォーマルメソッドが新規に入る余地があるとすれば，それは新しい概念のシステムを構築する時，あるいは新しい装置の開発ということになる．

むしろ，フォーマルメソッドを用いて，証明による検証を志向したシステムを提案できたらよいと

考えられる．例えば連動装置を例にとると，テーブル形式や結線図形式ではなく，電子連動装置導入当初に使われたシュプール図方式の方がむしろ定理証明向きであると考えられる．そこでは現状の連動装置の機能がすべて盛り込まれたわけではないが，より安全が確保できるのであれば採用するという発想があってもよい．この発想は見方を変えれば，ツールの制約に合わせてシステムを構築することを意味しており，その是非に関して議論が分かれるところではあるが，安全性を保証できるシステムや制御データ構造といった考えで，システムを構築できる提案ができれば，信号保安装置の制御や設計に関してパラダイムシフトを起こす可能性を秘めていると考える．

また，低レベルの処理プログラム，あるいはミドルウェアとして導入するというのももう 1 つの方向であると考えられる．ただし，鉄道用ソフトウェアのミドルウェアに何が必要であるかは，議論が熟していない．これについては，今後の議論の進展に合わせて展開していく必要がある．

なお，実際にフォーマルメソッドが信号保安装置に新規に導入される事例は新しいシステムのほうが入りやすいとしても，既存のシステムについても適用可能であることは示していく必要がある．また，国内では信号保安装置へのフォーマルメソッドの適用事例はほとんどないが，フォーマルメソッドの使用は IEC61508 でも推奨されていることであり，いざ適用の機運が生じたときに対応できるよう，前述したような参照モデルを構築しておくことは重要であり，地道にモデルを蓄積していく必要があるものと考えられる．

第 10 章

終わりに

本論文では定理証明手法の鉄道の信号保安装置への適用の検討を実施した．

ここで対象とする信号保安装置は，基本的な安全を担う装置である閉そく装置，連動装置，ATS，ATC といった範囲の装置であり，踏切制御装置に関しても応用可能である．これらの装置の処理の大部分は，数十〜数百 msec 毎に確定した入力を用いて論理演算を行うものである．部分的にはそれに加えて数値計算を行うが，ATS 信号や ATC 信号のアナログ波形を直接変復調する部分などを除くと，周期毎の入出力の変化が定義される誤差よりも小さいとみなせる範囲での計算となる．

これらの信号保安装置のうち，オフラインで検証可能な静的なデータと動的な制御部分についてケーススタディとしての検証を実施した．動的な部分に関しては，論理演算による制御部分と，論理演算には収まらない数値計算部分とで，それぞれケーススタディとしての検証を実施した．

静的な制御データに関しては，ディジタル ATC 線区データベースの仕様に関して，VDM-SL で記述されたデータベース仕様から生成される証明責務の証明を実施した．証明責務は全て証明できたが，この検証を通じて以下の知見を得た．

- 証明責務をながめ，反例を探してインタプリタで確かめることで仕様の誤りを見つけることができた．証明責務の自動生成は仕様の誤りを発見する上で重要であり，自動証明ができなくとも，反例を確かめる手法で仕様の品質向上が可能と考えられる．
- 自動証明ができれば，記述の誤りを見逃さずにすむ可能性がより高まる．これは仕様の品質をさらに向上させる．
- 証明を成功させるためには証明のテクニックを身につけることも必要だが，それ以上に仕様の理解が重要である．そのためには仕様をわかりやすくする努力も必要であり，単純かつしっかりとした構造を構築する必要がある．

リレー論理で記述可能な範囲の動的な制御として単線区間向け自動閉そく装置の検証，またリレー論理に収まらない数値計算部分としてブレーキ曲線計算プログラムの作成に B メソッドを適用した．単線区間向け自動閉そく装置の検証では，実際に行われる制御に関して検証を実施した．動的な条件についても事前事後の変数の関係を記述することによって，静的なデータと同様に検証ができることを示した．またブレーキ曲線計算については，論理計算に収まらない数値計算プログラムの証明済みコードの生成を行った．これにより数値計算についても定理証明技術の適用が可能で，テストだけでは得られないプログラムの高信頼化を図ることができたと評価できる．

また，この 2 つの事例を通じて，証明責務の生成に関して以下の知見を得た．

- 不変条件内や非決定的代入での論理和の使用，条件分岐の使用により証明責務の生成数が増大する．
- 同じことを別の表現で記述してしまうと自動証明できない証明責務が増大する．
- while ループを使用すると，そのループ不変条件に証明責務が増大する．
- 乗除算を用いると，中間値を含めて実装可能な範囲であることを確認するための証明責務が増大する．

これらの問題に対して，論理和の除去により証明責務を減らしたり，表明の使用により，証明責務の自動証明率を上げたりできることを示した．

以上により，信号保安装置の多くの部分について，装置仕様の検証モデルの作成が可能であり，検証の手間はかかるものの定理証明技術の適用が可能であることを確認できた．このことは本論文の主要な成果である．

また，前述のような定理証明技術適用にあたっての知見を多く得ることができた．これも本論文の成果である．

この技法が広く導入されるにはまだまだ時間がかかるとは考えられるが，制御データ検証への適用の可能性や，新しい装置への導入の可能性がある．既存システムの参照モデルの作成と併せて，信号保安装置の品質向上に資することができればよいと考えている．

謝辞

まずは，本論文をまとめるにあたり，博士課程入学の前からの 3 年間にわたり，終始にわたりましてご指導ご鞭撻を賜りました荒木啓二郎先生に，心より感謝を申し上げます．また，博士論文の副査として貴重なご意見を頂きました鵜林尚靖先生，興雄司先生，日下部茂先生におかれましてもこの場を借りて感謝申し上げます．特に日下部先生には 7 章の原論文を情報処理学会のプログラミング研究会で発表することをご提案くださいました．そこで発表した論文が情報処理学会の論文賞受賞となりました．さらに博士課程におけるアドバイザリー委員会において忌憚のない意見を頂戴しました北陸先端科学技術大学の青木利晃先生におかれましてもこの場を借りて感謝申し上げます．

この研究は，1999 年に私がデンマーク IFAD 社に派遣されて実施したことがきっかけとなっています．2 年余りの滞在中，形式手法について大いに勉強させてもらいましたし，荒木先生と知り合ったのも，デンマーク滞在中に参加した FM99 （Toulouse）においてであります．デンマーク渡航にあたり，IFAD 社への派遣をお膳立てくださいました秋田雄志（当時鉄道総合技術研究所理事，元同理事長，現同フェロー），荻野隆彦（当時鉄道総研輸送システム開発推進部長，現一般財団法人研友社理事長），平尾裕司（当時鉄道総研列車制御研究室主任技師，現長岡技術科学大学教授）の各氏，また上司として快く私を送り出して下さいました高重哲夫（当時鉄道総研信号研究室主幹技師，現ジェイアール総研電気システム社長），渡辺郁夫（同主任技師，現鉄道総合技術研究所理事）の各氏には改めて感謝申し上げます．

IFAD 社では社長の Benny Graff Mortensen 氏をはじめ Peter Gorm Larsen（現オーフス大学教授）, Sten Agerholm，Kim Sunesen の各氏に大変お世話になりました．5，6 章の内容のうち，帰国後に実施した連動仕様の日本語化を除く大部分は，3 人に指導してもらったものです．5 章については，課題を通じて Peter Gorm Larsen 氏に VDM における仕様記述の考え方を色々と教わりました．また，CORBA API による GUI の作成を提案したのは Peter Gorm Larsen 氏です．6 章の証明ツールの主要な部分は Kim Sunesen 氏の手によります．公私にわたる付き合い，そしてそれを含むデンマークでの 2 年余りの生活はその後の私の人生に大きな影響を与えています．ここで日本語で書いても直接伝わる話ではないのですが，この場を借りて感謝申し上げます．

5，6 章の題材の大部分は元々は鉄道総研の先輩である福田光芳氏（現在鉄道総研列車制御研究室長）が中心となって記述したものを改良したものです．福田氏には 5，6 章の内容の実施に当たって，元々の仕様の考え方について示唆をいただいたほか，海外生活当初には色々と悩みなどを聞いてもらい，相当助けられました．改めて感謝申しあげます．

8 章のブレーキ曲線の計算については，2005 年前後に荻野隆彦，高橋一裕（日本信号（株））の各氏と B メソッドを勉強する中で検討したことがベースとなっています．VDM の代わりに証明をするために B を使うことにしたのは荻野氏との議論の結果です．荻野氏におかれましては，先のデン

マーク派遣だけでなく，帰国後においても，いろいろと形式手法適用について議論させてもらったことに改めて感謝します．

また，社会人としての博士後期課程での研究に関し，鉄道総研において上司として色々とご支援下さいました土屋隆司（旧信号・情報技術研究部長），平栗滋人（現信号・情報技術研究部長），新井英樹（信号システム研究室長）の各氏，鉄道総研内で形式手法の議論をいろいろさせてもらった遠山喬君，丹羽順一君におかれましても感謝申し上げます．

最後に，デンマーク渡航を喜んでくれた亡き父，その父の遺志だからだと博士後期課程進学を支援してくれた母には大変感謝しております．父の病気を知ったのはデンマーク渡航後で，私に心配をかけまいと癌にかかっていたことを伏せていたのであります．デンマークに私を訪ねることを楽しみにしていたのに，ついにそれが叶わなかったのは今でも心残りです．父が亡くなったのはデンマーク滞在中であり，滞在が終了するまでの１年弱の間，そして帰国後も母や弟にはいろいろと迷惑をかけてしまいしたが，論文をここまでこぎつけられたのは両親の応援があったからだと思っています．ありがとうございました．

参考文献

[1] P. G. Larsen and J. Fitzgerald. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, 1998.

[2] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[3] J.-R. Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 1998.

[4] 奥村幾正, 渡辺俊勝. フェイルセイフ形電子計算機を用いた鉄道信号用連動装置に関する研究. 鉄道技術研究報告, No. 803, 1972.

[5] D. A. Anderson and G. Metze. Design of Totally Self-Checking Check Circuit for m-out-of-n Codes. *IEEE Transactions on Computer*, Vol. C-22, No. 3, pp. 263–269, 1973.

[6] J. E. Smith and G. Metze. Strongly Fault Secure Logic Networks. *IEEE Transactions on Computer*, Vol. C-27, No. 6, pp. 491–499, 1978.

[7] 南谷崇, 河村俊明. セルフチェッキングシステムにおける誤り安全と誤り伝搬性の概念. 電子情報通信学会論文誌, Vol. J68-D, No. 12, pp. 2007–2014, 1985.

[8] 南谷崇, 河村俊明. セルフチェッキングプロセッサの構成法. 電子情報通信学会論文誌, Vol. J68-D, No. 12, pp. 2015–2025, 1985.

[9] 森田悦夫, 秋田雄志, 宮崎孝俊, 篠原勝雄, 関口晋, 鈴木紀夫, 野田耕一郎. 東神奈川駅電子連動装置のシステム概要. 鉄道における国際サイバネティクス利用国内シンポジウム論文集, pp. 417–421, 1985.

[10] 中村英夫, 西堀典幸, 塩沢一雄, 瀬戸通夫, 高橋昇. 電子連動装置（SMILE）のハードウェア. 鉄道における国際サイバネティクス利用国内シンポジウム論文集, pp. 422–426, 1985.

[11] 高重哲夫. 100 系 ATC 車上受信装置. 信号保安, Vol. 40, No. 5, pp. 211–214, 1985.

[12] 岡田恭一. デジタル ATC の開発と導入. *JR East Technical Review*, pp. 27–30, 2003.

[13] 渡辺俊勝, 播磨義憲, 安部正夫, 梅津篤, 宗方江一郎. 電子連動装置（SMILE）のハードウェア. 鉄道における国際サイバネティクス利用国内シンポジウム論文集, pp. 427–431, 1985.

[14] 森川俊紀, 遠藤忠義, 川谷文夫, 荒伸幸. 結線入力方式による新しい電子連動装置（K-5 形）. 京三サーキュラー, Vol. 42, No. 1, pp. 19–29, 1991.

[15] 北原文夫, 上條恵司, 川岸紀夫, 解良和郎, 戸次圭介. 軌道回路予約論理に基づく鉄道用線路保守作業管理システムの開発. 電気学会論文誌, Vol. 120-D, No. 1, pp. 126–135, 2000.

[16] 鉄道総合技術研究所. 列車保安制御システムの安全性技術指針, 1996.

[17] J. Bicarregui (ed.). *Proof in VDM : case studies*. Springer-Verlag, 1998.

[18] VDM information web site, http://vdmtools.jp.

[19] Overture Tool, http://overture.org.

[20] CORBA, information available at http://www.corba.org.

[21] J. Bicarregui, J. Fitzgerald, P. Lindsay, R. Moore, and B. Ritchie. *Proof in VDM : A Practitioner's Guide*, Springer-Verlag, 1994.

[22] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural : A Formal Development Support System*, Springer-Verlag, 1993.

[23] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur (ed.), *11th International Conference on Automated Deduction (CADE)*, LNAI 607, pp. 748–752, Springer-Verlag, 1992.

[24] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828, Springer-Verlag, 1994.

[25] A. Hall. Using Formal Methods to Develop an ATC Information System. *IEEE Software*, Vol. 13, No. 2, pp. 66–76, 1996.

[26] T. Clement, I. Cottam, P. Froome, and C. Jones. The Development of a Commercial "Shrink-Wrapped Application" Safety Integrity Level 2: The DUST-EXPERT. Story. In M. Felici, K. Kanoun, and A. Pasquini (eds.), *Proceedings of SAFECOMP'99*, LNCS 1698, pp. 216–225, Springer-Verlag, 1999.

[27] M. v. d Berg, M. Verhoef, and M. Wigmans. Formal Specification of Auctioning System Using VDM++ and UML, an Industrial Usage Report. In J. Fitzgerald and P. G. Larsen (eds.), *Proceedings of VDM workshop 99 (VDM in Practice)*, pp. 67–79, 1999.

[28] P. R. Smith and P. G. Larsen. Applications of VDM in Banknote Processing. In J. Fitzgerald and P. G. Larsen (eds.), *Proceedings of VDM workshop 99 (VDM in Practice)*, pp. 67–79, 1999.

[29] 佐原伸. 事務システムにおける形式仕様適用例. ソフトウェア・シンポジウム, 2001.

[30] 栗田太郎. フォーマルメソッドの新潮流：Part II:産業界への応用: 3. 携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用. 情報処理学会誌, Vol. 49, No. 5, pp. 16–23, 2008.

[31] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1991.

[32] Atelier B, information available at http://www.atelierb.eu.

[33] M. Leuschel and M. Butler. ProB: A model checker for B. In K. Araki, S. Gnesi, and D. Mandrioli (eds.), *Proceedings of FME 2003*, LNCS 2805, pp. 855–874, Springer-Verlag, 1991.

[34] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor: A Successful Application of B in a Large Project. In J. Wing, J. Woodcook, and J. Davies (eds.), *Proceedings of FM'99*, LNCS 1708, Vol.I, pp. 369–387, Springer-Verlag, 1999.

[35] F. Badeau and A. Amelot. Using B as a High Level Programming Language in an Industrial Project: Roissy VAL. In H. Treharne et al. (eds.), *Proceedings of ZB2005*, LNCS 3455, pp. 334–354, Springer-Verlag, 2005.

[36] T. Lecomte. Safe and Reliable Metro Platform Screen Doors Control/Command Systems. In J. Cuellar and T. Maibaum (eds.), *Proceedings of FM2008*, LNCS 5014, pp. 430–434, Springer-Verlag, 2008.

[37] B. Gomes, D. Déharbe, A. Moreira, and K. Moraes. Applying the B Method for the Rigorous Development of Smart Card Applications. In M. Frappier et al. (eds.), *Proceedings of ABZ2010*, LNCS 5977, pp. 203–216, Springer-Verlag, 2010.

[38] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[39] Event-B.org webpage, http://www.event-b.org/index.html.

[40] M. Butler and I. Maamria. Practival Theory Extension in Event-B. In *Theories of Programming and Formal Methods*, LNCS 8051, pp. 67–81, Springer-Verlag, 2013.

[41] D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin. SMT Solvers for Rodin. In J. Derrick et al. (eds.), *Proceedings of ABZ2012*, LNCS 7316, pp. 194–207, Springer-Verlag, 2012.

[42] D. Merý and N. K. Singh. Automatic code Generation from event-B models. In *Proceedings of SoICT'11*, pp. 179–188. ACM, 2011.

[43] J.-R. Abrial, W. Su, and H. Zhu. Formalizing Hybrid Systems with Event-B. In J. Derrick et al. (eds.), *Proceedings of ABZ2012*, LNCS 7316, pp. 178–193, Springer-Verlag, 2012.

[44] 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, jun 2013.

[45] D. Sabatier, L. Burdy, A. Requet, and J. Guéry. Formal Proofs for the NYCT Line 7 (Flushing) Modernization Project. In J. Derrick et al. (eds.), *Proc. ABZ2012*, LNCS 7316, pp. 369–372, Springer-Verlag, 2012.

[46] Isabelle/HOL-Z home page, https://www.brucker.ch/projects/hol-z/.

[47] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *Definitions of Standard ML (Revised)*. The MIT Press, 1997.

[48] G. Klein and M. Wildmoser. Verified Bytecode Subroutines. In D. Basin and B. Wolff (eds.), *Proceedings of TPHOLs 2003*, LNCS 2758, pp. 55–70, 2003.

[49] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification OS Kernel. In *22nd ACM Symposium on Operating System Principles*, pp. 200–207, 2009.

[50] G. Gonthier. Formal Proof – The Four Color Theorem. *Notices of AMS*, Vol. 55, No. 11, pp. 1370–1414, 2008.

[51] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, Vol. 52, No. 7, pp. 107–115, 2009.

[52] S. Owre, J. M. Rushby, N. Shankar, and D. Stringer-Calvert. PVS Prover Guide, available at http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf, 2001.

[53] S. Owre, J. M. Rushby, N. Shankar, and D. Stringer-Calvert. PVS: An Experience Report. In D. Hutter, W. Stephan, P. Traverso, and M. Ullman (eds.), *Applied Formal Methods—*

FM-Trends 98, LNCS 1641, pp. 338–345, Springer-Verlag, 1998.

[54] V. Carreno, H. Gottliebsen, R. Butler, and S. Kalvala. Formal Modeling and Analysis of a Preliminary Small Aircraft Transportation System (SATS) Concept. Technical memorandum, NASA/TM-2004-219999, NASA, 2004.

[55] G. J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.

[56] K. L. McMillan. *Symbolic Model Checkiing*. Kluwer Academic Publishers, 1993.

[57] A. E. Haxthausen, A. A. Kjær, and M. L. Bliguet. Formal Development of a Tool for Automated Modelling and Verification of Relay Interlocking Systems. In M. Butler and W. Schulte (eds.), *Proceedings of FM2011*, LNCS 6664, pp. 118–132, Springer-Verlag, 2011.

[58] A. Cimatti, E. Clarke, F. Giunhiglia, and M. Roveri. NuSMV : a new Symbolic Model Verifier. In N. Halbwachs and D. Peled (eds.), *Proceedings of CAV'99*, LNCS 1633, pp. 495–499, Springer-Verlag, 1999.

[59] A. Cimatti, E. Clarke, E. Giunhiglia, F. Giunhiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2 : An OpenSource Tool for Symbolic Model Checking. In G. Berry, H. Comon, and A. Finkel (eds.), *Proceedings of CAV2002*, LNCS 2404, pp. 359–364, Springer-Verlag, 2002.

[60] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic Model Checking for Sequential Circuit Verification. *IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 4, pp. 401–422, 1994.

[61] A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying Safety Properties of a PowerPC Microprocessor Using Symbolic Model Checking. In N. Halbwachs and D. Peled (eds.), *Proceedings of CAV'99*, LNCS 1633, pp. 60–71, Springer-Verlag, 1999.

[62] R. J. Anderson, P. Beame, S. Burns, W. Chan, F. Modugno, D. Notkin, and J. D. Reese. Model Checking Large Software Specifications. In D. Garlan (ed.), *Proceedings of SOFT96 SIGSOFT '96*, pp. 156–166. ACM, 1996.

[63] 亀山幸義, 中島一. ケーススタディ：モデル検査と定理証明を用いた鉄道信号制御システムの検証. シンポジウム「システム検証の科学技術」予稿集, pp. 82–91, 2004.

[64] G. J. Holzmann and R. Joshi. Model-Driven Software Verification. In S. Graf and L. Mounier (eds.), *Proceedings of SPIN 2004*, LNCS 2989, pp. 76–91, Springer-Verlag, 2004.

[65] NASA Engineering and Safety Center. Technical Assessment Report: National Highway Traffic Safety Administration (NHTSA), Toyota Unintended Acceleration Investigation, Appendex A : Software, 2011. available at http://www.nhtsa.gov/staticfiles/nvs/pdf/NASA-FR_Appendix_A_Software.pdf.

[66] 川村正, 藤井英明, 土田勝紀, 高橋和子. 鉄道信号システムの連動装置の形式的検証向けモデル化と検証環境構築. 電子情報通信学会論文誌, Vol. J88-D-I, No. 12, pp. 1727–1739, 2005.

[67] 継電連動装置標準結線図（ＪＲグループ監修）. 日本鉄道電気技術協会, 1987.

[68] T. Ball and S. K. Rajamani. The SLAM Project : Debugging System Software via Static Analysis. In *Proceedings of POPL '02*, pp. 1–3, 2002.

[69] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model Chekcing Programs.

Automated Software Engineering, pp. 203–232, 2003.

[70] SCADE, information available at http://www.esterel-technologies.com/.

[71] G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, Vol. 19, No. 2, pp. 87–152, 1992.

[72] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Data Flow Programming Language LUSTRE. *Proceedings of the IEEE*, Vol. 79, No. 9, pp. 1305–1320, 1991.

[73] Mathworks 日本語ページ, http://jp.mathworks.com/product.

[74] Prover Plugin, information available at http://www.prover.com/products/prover_plugin.

[75] S. Dajani-Brown, D. Cofer, and A. Bouali. Formal Verification of an Avionics Sensor Voter Using SCADE. In Y. Lakhnech and S. Yovine (eds.), *Proceedings of FORMATS/FTRTFT 2004*, LNCS 3253, pp. 5–20, Springer-Verlag, 2004.

[76] I. Daskaya, M. Huhn, and S. Milius. Formal Safety Analysis in Industrial Practice. In G. Salaün and B. Schätz (eds.), *Proceedings of FMICS 2011*, LNCS 6959, pp. 68–94, Springer-Verlag, 2011.

[77] 足立正和, 富永孝, 佐野範佳, 潮俊光. 同期型言語を用いたソフトウェア検証：クルーズコントロールシステムにおける事例紹介. コンピュータソフトウェア, Vol. 23, No. 3, pp. 85–90, 2006.

[78] 小西晃輔, 高野求, 島添敏之. 形式手法によるソフトウェア開発の試みと鉄道信号アプリケーションへの適用可能性. 形式手法の産業界応用ワークショップ 2011 予稿集, pp. 65–78, 2011.

[79] A. Borälv. Case Study: Formal Verification of a Computerised Railway Interlocking. *Formal Aspects of Computing*, Vol. 10, pp. 338–360, 1998.

[80] 平尾祐司他. 鉄道連動装置の機能仕様記述へのフォーマルメソッドの適用. 平成 8 年電気学会全国大会予稿集, 1996.

[81] PROSPER project, information available at http://www.dcs.gla.ac.uk/prosper/.

[82] 高重哲夫, 渡辺郁夫. 高速高密度区間用ディジタル ATC の開発. 鉄道総研報告, Vol. 9, No. 1, pp. 49–54, 1995.

[83] 福田光芳, 渡辺郁夫, 平尾裕司. 鉄道分野での高信頼性データベースの設計に関する一考察. 電子情報通信学会 FTS 研究会, 1997.

[84] S. Agerholm and K. Sunesen. Formalizing a Subset of VDM-SL in HOL. Technical report, IFAD, 1999. available at http://overture.org.

[85] B. K. Aichernig and P. G. Larsen. A Proof Obligation Generator for VDM-SL. In J. Fitzgerald, C. Jones, and P. Lucas (eds.), *Proceedings of FME'97*, LNCS 1313, pp. 338–357, Springer-Verlag, 1997.

[86] S. Agerholm and K. Sunesen. Reasoning about VDM-SL Proof Obligations in HOL. Technical report, IFAD, 1999. available at http://overture.org.

[87] The RAISE Method Group. *The RAISE Development Method*. BCS Practitioner Series. Prentice Hall, 1995.

[88] 継電連動装置（CTC 関連および ARC）標準結線図（改訂版）. 日本鉄道電気技術協会, 2000.

[89] 閉そく装置. 鉄道技術者のための電気概論. 日本鉄道電気技術協会, 1993.

[90] M. Leuschel, J. Falampin, F. Fritz, and D. Plagge. Automated Property Verification for Large Scale B Models. In *Proceedings of FM 2009*, LNCS 5850, Springer-Verlag, 2009.

[91] 寺田夏樹. SMT ソルバによる単線自動閉そく装置の検証. 電子情報通信学会技術報告, Vol. 110, No. 362, pp. 31–36, 2012.

[92] A. E. Haxthausen, J. Peleska, and S. Kinder. A formal approach for the construction and verification of railway control system. *Formal Aspects of Computing*, Vol. 23, No. 2, pp. 191–219, 2011.

[93] J. Bendisposto and M. Leuschel. Proof Assisted Model Checking for B. In K. Breitman and A. Cavalcanti (eds.), *Proceedings of ICFEM 2009*, LNCS 5885, Springer-Verlag, 2009.

[94] D. Mentré, C. Marché, J.-C. Filliâtre, and M. Asuka. Discharging Proof Obligations from Atelier B using Multiple Automated Provers. In J. Derrick et al. (eds.), *Proceedings of ABZ2012*, LNCS 7316, pp. 238–251, 2012.

業績まとめ

形式手法関連

原著論文

- 寺田夏樹, 福田光芳． 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, pp. 33–38, 2002.
- N. Terada, M. Fukuda. Application of Formal Methods to the Railway Signalling Systems. Quaterly Report of RTRI, Vol. 43, No. 4, pp. 169–174, 2002.
- 寺田夏樹．段階的詳細化手法に基づく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.
- N. Terada. Application of Formal Methods to Signalling Systems Using Techniques Based on Stepwise Refinement. Quarterly Report of RTRI, Vol. 49, No. 3, pp. 168–172, 2008.
- 寺田夏樹，遠山喬．信号設備の安全性に関する仕様検証手法の適用検討．鉄道総研報告，Vol. 27, No. 2, pp. 35–40, 2013.
- N. Terada, T. Toyama. Application of Verification Methods to Specifications of Signalling Equipment. Quarterly Report of RTRI, Vol. 54, No. 4, pp. 202–207, 2013.
- 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.
- 寺田夏樹．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．情報処理学会論文誌ジャーナル，Vol. 56, No. 4，pp. 1278–1291, 2015.

国際会議

- N. Terada. Formal Integrity Analysis of Digital ATC Track Database. Proceedings of World Congress on Railway Research(WCRR), Cologne, 2001.11.
- N. Terada, Integrity Analysis of Digital ATC Track Database with Automatic Proofs, Proceedings of VDM workshop, FM2002, Copenhagen, 2002.7.
- N. Terada. Application of Formal Methods to Automatic Train Control Systems, Proceeding of Symposium on Formal Methods for Railway Operation and Control Systems (FORMS), pp. 143–148, Budapest, 2003.5.

解説記事

- 高橋一裕, 林俊範, 中山利宏, 寺田夏樹. 鉄道信号システムへのフォーマルメソッドの適用. 鉄道と電気技術, Vol. 20, No. 2, pp. 28–32, 2009.
- 寺田夏樹. VDM の紹介と鉄道信号への適用例. 日本信頼性学会誌, Vol. 34. No. 8, pp. 384–389, 2012.

国内口頭発表

- 寺田夏樹, 福田光芳. 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, 2001.12.
- 寺田夏樹. ディジタル ATC 線区データベースの証明による検証. クリティカルソフトウェアワークショップ 2002, つくば，2002.3 （予稿なし）.
- 寺田夏樹. 段階的詳細化によるシステムの高信頼化手法. 第 5 回クリティカルソフトウェアワークショップ, pp. 193–199, 東京，2005.11.
- 寺田夏樹, 高橋一裕, 荻野隆彦. 段階的詳細化に基づくシステムの高信頼化手法. 信学技報, Vol. 105, No. 458, DC2005-65, pp. 1–6, 岩国, 2005.12.
- 寺田夏樹. 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. 信学技報, Vol. 108, No. 243, R2008-32, pp. 27–32, 北九州，2008.10.
- 寺田夏樹. B メソッドによる単線自動閉そく装置の検証. 信学技報, Vol. 109, No. 334, DC2009-61, pp. 31–36, 松江，2009.12.
- 寺田夏樹. 単線自動閉そく装置の B メソッドによるモデル化と検証. 第 17 回鉄道技術連合シンポジウム（J-RAIL2010）講演論文集, pp. 433–436, 東京, 2010.12.
- 寺田夏樹. モデル検査法による単線自動閉そく装置の検証. 信学技報, Vol. 110, No. 333, DC2010-57, pp.31–35, 米子, 2010.12.
- 寺田夏樹. SMT ソルバによる単線自動閉そく装置の検証. 信学技報, Vol. 112, No. 362, DC2012-79, pp. 31–36, 福井, 2012.12.
- 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. 情報処理学会プログラミング研究会，豊洲, 2013.11.

形式手法以外

原著論文

- 寺田夏樹, 篠田裕之, 安藤繁. 多モード音響共鳴を用いたテンソルセル触覚センサ. 計測自動制御学会論文集, Vol. 33, No. 4, pp. 234–240, 1997.
- 福田光芳, 渡辺郁夫, 寺田夏樹, 島添敏之, 奥谷民雄. 要求分析と統合的ライフサイクルコスト評価に基づいた鉄道信号システム構築手法の検討. 電気学会論文集 D, Vol. 125, No. 7, pp.

681–690, 2005.

- 奥谷民雄, 中村信幸, 寺田夏樹, 福田光芳, 舘裕, 稲田聡, 伊藤秀憲, 若尾真治．山岳トンネルの電磁遮蔽効果に対する解析手法の高度化． 電気学会論文集 D, Vol. 127, No. 4, pp. 391–399, 2007.
- 奥谷民雄, 中村信幸, 寺田夏樹, 舘裕, 稲田聡, 小川知行, 孫佳男, 若尾真治. 鉄道高架橋の電磁遮蔽効果に対する解析手法の高度化． 電気学会論文集 D, Vol. 128, No. 3, pp. 310–320, 2008.
- 横田倫一, 須貝孝博, 小山智之, 遠藤博昭, 葛西隆也, 武田真吾, 寺田夏樹, 宮本真行．北陸新幹線（高崎・金沢間）60Hz き電区間対応 DS-ATC の開発．電気学会論文集 D，Vol. 132, No. 2, pp. 1–10, 2012.

国際会議

- N. Terada, M. Fukuda, I. Watanabe, T. Okutani. New Strategy for Configuring Signalling Systems. Proceedings of ASPECT 2003, pp. 115–119, London, 2003.
- M. Fukuda, N. Terada, T. Ban. A New Approach to Prevent Shunting Malfunction of Track Circuits Based on Quantified Electric Resistance between Rail and Wheel. Proceedings of ASPECT 2008, London, 2008.

国内口頭発表（本人発表分のみ）

- N. Terada, Y. Kumano, H. Honma, H. Shinoda and S. Ando. Acoustic Resonant Tactile Sensing. Proceedings of 13th Sensor Symposium, Tokyo, pp. 221–224, 1995.
- 寺田夏樹, 熊野洋, 本間啓人, 篠田裕之, 安藤繁. 空洞の共鳴を用いる触覚センシング. 第 35 回計測自動制御学会学術講演会予稿集, 札幌，pp. 615–616, 1995.
- 寺田夏樹, 福田光芳, 渡辺郁夫, 島添敏之, 中村信幸, 奥谷民雄. 信頼性分析に基づいた信号設備構成案の評価. 平成 15 年電気学会産業応用部門大会, 東京， 2003.8.
- 寺田夏樹, 福田光芳, 渡辺郁夫, 奥谷民雄. 整備新幹線向け信号設備の機能仕様の分析. 電気学会交通・電気鉄道研究会, TER-05-41, 東京，2005.9.
- 寺田夏樹, 福田光芳. 長大軌道回路の耐ノイズ性能向上策. 電気学会交通・電気鉄道研究会, TER-09-8, 名古屋，2009.2.
- 寺田夏樹, 福田光芳. 簡易な符号化による長大軌道回路の耐ノイズ性能向上. 平成 22 年電気学会産業応用部門大会, 東京，2010.8.
- 寺田夏樹, 福田光芳. 長大軌道回路の耐ノイズ性能向上策. 第 47 回鉄道サイバネ・シンポジウム, 東京，2010.11.

解説記事

- 寺田夏樹. デジタル ATC. 鉄道と電気技術，Vol. 9, No. 16, pp. 16–17, 1998.
- 長谷川敏明，寺田夏樹，大津光雄，加藤利幸．信号ケーブル障害点距離検知器の開発．鉄道と

電気技術，Vol. 16, No. 9, pp. 24–28, 2005.

- 奥谷民雄，中村伸幸，寺田夏樹，福田光芳，舘裕，稲田聡．山岳トンネルの電磁遮蔽効果に対する解析手法の高度化．鉄道と電気技術，Vol. 19, No. 12, pp. 51–62, 2008.
- H. Yamamoto, S. Hiraguri, N. Terada, S. Shiomi. Signalling Systems Suitable for Secondary Lines, Japanese Railway Engineering, Vol. 51, No. 3, pp.11–13, 2011.
- 寺田夏樹，兎束哲夫. 講座－電車線路設備について (17). 鉄道と電気技術, Vol. 24, No. 2, pp. 72–75, 2013.
- 廿日出悟，寺田夏樹．車両主回路からの誘導障害．鉄道と電気技術, Vol. 24, No. 10, pp. 18–22, 2013.

鉄道総研報告および Quartarly Report of RTRI（QR）

- 高重哲夫，渡辺郁夫，福田光芳，寺田夏樹．中国高速鉄道用ディジタル ATC の開発．鉄道総研報告，Vol. 13, No. 8, pp. 13–20, 1999.
- 福田光芳，寺田夏樹，中島一歩，渡辺郁夫．軌道回路の列車検知性能向上に関する研究．鉄道総研報告，Vol. 16, No. 7, pp. 15–20, 2002.
- 福田光芳，渡辺郁夫，寺田夏樹，島添敏之，奥谷民雄．アベイラビリティ評価指標による鉄道信号システム開発手法．鉄道総研報告 Vol. 18, No. 7, pp. 17–22, 2004.
- 長田実，渡邉朝紀，中村英男，寺田夏樹．車載装置による軌道短絡性能の向上．鉄道総研報告，Vol. 20, No. 7, pp. 23–28, 2006.
- 福田光芳，板垣朋範，寺田夏樹．軌道回路の短絡不良要因と改善手法．鉄道総研報告，Vol. 21, No. 11, pp. 5–10, 2007.
- 福田光芳，古川信幸，寺田夏樹，須貝孝博．無絶縁軌道回路に対応した新幹線用デジタル ATC の開発．鉄道総研報告，Vol. 21, No. 11, pp. 11–16, 2007.
- M. Fukuda, N. Terada, T. Ban. Study on Quantifying and Reducing Electrical Resistance between Wheels and Rails, Quaterly Report of RTRI, Vol. 49, No. 3, pp. 158–162, 2008.
- 寺田夏樹．電気の流れ方で列車を検知する．RRR, Vol. 66, No. 10, pp. 34–37, 2009.
- 寺田夏樹，福田光芳．簡易な符号伝送による低周波軌道回路の耐ノイズ性能向上．鉄道総研報告，Vol. 24, No. 3, pp. 11–16, 2010.
- 福田光芳，寺田夏樹，北野公一，遠山喬．耐ノイズ性を向上した中間軌道回路の開発．鉄道総研報告，Vol. 25, No. 5, pp. 17–22, 2011.
- 寺田夏樹，横田倫一，須貝孝博，葛西隆也，武田真吾．北陸新幹線 50/60Hz き電両用区間対応 DS-ATC の開発．鉄道総研報告，Vol. 26, No. 7, pp. 17–22, 2012.
- 遠山喬, 福田光芳, 大和田厚祐, 寺田夏樹, 原智昭, 本多秀行．車輪・レール接触状態に基づく短絡抵抗推定手法．鉄道総研報告，Vol. 28, No. 4, pp. 5–10, 2014.
- 寺田夏樹．鉄道技術　来し方行く末　軌道回路．RRR, Vol. 71, No. 5, pp. 28–31, 2014.
- 寺田夏樹，赤木雅陽，横田倫一．電気設備の境界：交流 50/60Hz セクションを高速で通過する．RRR, Vol. 71, No. 9, pp. 20–23, 2014.
- 新井英樹，寺田夏樹. 信号保安装置の動向．RRR, Vol. 72. No. 2, pp. 24–27, 2015.

付録 A

連動装置の VDM モデル

```
types
進路名 = token;
軌道回路名 = token;
転てつ器名 = token;

信号機の状態 = <進行> | <停止>;
転てつ器の向き = <定位> | <反位>;
軌道回路の状態 = <在線> | <非在線>;
鎖錠 = <鎖錠> | <解錠>;
進路の状態 = <未設定> | <設定> | <鎖錠中>;

進路条件 :: 転てつ器 : map 転てつ器名 to 転てつ器の向き
            鎖錠進路 : set of 進路名
            信号制御 : set of 軌道回路名
            進路鎖錠 : set of 軌道回路名;

連動図表 :: 進路 : map 進路名 to 進路条件
            転てつ器 : map 転てつ器名 to set of 軌道回路名
            軌道回路 : set of 軌道回路名
inv mk_連動図表 (進路, 転てつ器, 軌道回路) ==
    (forall r in set dom 進路 &
      dom 進路 (r).転てつ器 subset dom 転てつ器 and
      dom 進路 (r).鎖錠進路 subset dom 進路 and
      dom 進路 (r).信号制御 subset 軌道回路 and
      dom 進路 (r).進路鎖錠 subset 軌道回路) and
    forall p in set dom 転てつ器 & 転てつ器 (p) subset 軌道回路;

state 連動装置 of
       連動 : 連動図表
       軌道回路 : map 軌道回路名 to 軌道回路の状態
       転てつ器 : map 転てつ器名 to 転てつ器の向き
       転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
       信号機 : map 進路名 to 信号機の状態
       進路設定 : map 進路名 to 進路の状態

```

```
36  inv mk_連動装置(連動, 軌道回路, 転てつ器, 転てつ器の鎖錠, 信号機, 進路設定) ==
37      dom 軌道回路 = 連動.軌道回路 and
38      dom 転てつ器 = dom 連動.転てつ器 and
39      dom 転てつ器の鎖錠 = dom 連動.転てつ器 and
40      dom 信号機 = dom 連動.進路 and
41      dom 進路設定 = dom 連動.進路 and
42      てっ査鎖錠(軌道回路, 連動.転てつ器, 転てつ器の鎖錠) and
43      forall r in set dom 進路設定 &
44        (信号機(r) = <進行> => 進路設定(r) = <設定>) and
45        (進路設定(r) = <設定> => 進路開通(進路設定, 連動.進路(r), 転てつ器, 転てつ器の鎖錠)) and
46        (信号機(r) = <進行> =>
47           forall tc in set 連動.進路(r).信号制御 & 軌道回路(tc) = <非在線>) and
48        (信号機(r) = <停止> =>
49           exists tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <在線>)))
50
51  init 連動 == 連動 = mk_連動装置(IL,
52       {TC1 |-> <非在線>, TC2 |-> <非在線>, TC3 |-> <非在線>, TC4 |-> <非在線>},
53       {P1 |-> <定位>, P2 |-> <定位>}, {P1 |-> <解放>, P2 |-> <解放>},
54       {R1 |-> <停止>, R2 |-> <停止>, R3 |-> <停止>, R4 |-> <停止>, R5 |-> <停止>},
55       {R1 |-> <未設定>, R2 |-> <未設定>, R3 |-> <未設定>,
56       R4 |-> <未設定>, R5 |-> <未設定>})
57  end
58
59  functions
60  てっ査鎖錠 : (map 軌道回路名 to 軌道回路の状態) * (map 転てつ器名 to set of 軌道回路名) *
61               (map 転てつ器名 to 鎖錠) -> bool
62  てっ査鎖錠(軌道回路, 鎖錠条件, 転てつ器の鎖錠) ==
63      forall tc in set dom 軌道回路状態, p in set dom 鎖錠条件 &
64        (tc in set 鎖錠条件(p) and 軌道回路(tc) = <在線>) => 転てつ器の鎖錠(p) = <鎖錠>
65  pre dom 転てつ器の鎖錠 = dom 鎖錠条件 and
66      forall tcs in set rng 鎖錠条件 & tcs subset dom 軌道回路状態;
67
68  進路開通 : (map 進路名 to 進路の状態) * 進路条件 * (map 転てつ器名 to 転てつ器の向き) *
69             (map 転てつ器名 to 鎖錠)  -> bool
70  進路開通(進路状態, 進路成立条件, 転てつ器向き, 転てつ器の鎖錠) ==
71      (forall p in set dom 進路成立条件.転てつ器 &
72          転てつ器の鎖錠(p) = <鎖錠> and
73          進路成立条件.転てつ器(p) = 転てつ器向き(p)) and
74      (forall r1 in set 進路成立条件.鎖錠進路 & 進路状態(r1) = <鎖錠中>)
75  pre dom 進路成立条件.転てつ器 subset dom 転てつ器向き and
76      dom 進路成立条件.転てつ器 subset dom 転てつ器の鎖錠 and
77      進路成立条件.鎖錠進路 subset dom 進路状態;
78
79  operations
80  進路設定要求(r : 進路名)
81  ext rd 連動 : 連動図表
82      rd 軌道回路 : map 軌道回路名 to 軌道回路の状態
83      wr 転てつ器 : map 転てつ器名 to 転てつ器の向き
```

```
84      wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
85      wr 信号機 : map 進路名 to 信号機の状態
86      wr 進路設定 : map 進路名 to 進路の状態
87  pre r in set dom 進路設定 and
88      進路設定 (r) = <未設定> and
89      (forall r1 in set 連動. 進路 (r). 鎖錠進路 & 進路設定 (r1) <> <設定>) and
90      (forall p in set dom 連動. 進路 (r). 転てつ器 &
91          転てつ器 (p) = 連動. 進路 (r). 転てつ器 (p) or 転てつ器の鎖錠 (p) = <解放>) and
92      (forall tc in set 連動. 進路 (r). 信号制御 &  軌道回路 (tc) = <非在線>)
93  post 進路設定 = 進路設定~ ++ {r |-> <設定>} ++
94                  {r1 |-> <鎖錠> | r1 in set 連動. 進路 (r). 鎖錠進路} and
95       (let 転換する転てつ器 =
96          {p | p in set dom 連動. 進路 (r). 転てつ器 & 転てつ器の鎖錠~(p) = <解放>} in
97          転てつ器の鎖錠 = 転てつ器の鎖錠~ ++ {p |-> <鎖錠> | p in set 転換する転てつ器} and
98          転てつ器 = 転てつ器~ ++
99                     {p |-> 連動. 進路 (r). 転てつ器 (p) | p in set 転換する転てつ器}) and
100      信号機 = 信号機~ ++ {r |-> <進行>};
101
102 進路設定要求 l : 進路名 ==> ()
103 進路設定要求 l (r) ==
104    let 転換する転てつ器 =
105           {p | p in set dom 連動. 進路 (r). 転てつ器 & 転てつ器の鎖錠 (p) = <解放>} in
106    (進路設定 := 進路設定 ++ {r |-> <設定>} ++
107           {r1 |-> <鎖錠中> | r1 in set 連動. 進路 (r). 鎖錠進路};
108     転てつ器の鎖錠 := 転てつ器の鎖錠 ++ {p |-> <鎖錠> | p in set 転換する転てつ器};
109     転てつ器 := 転てつ器 ++ {p |-> 連動. 進路 (r). 転てつ器 (p) | p in set 転換する転てつ器};
110     信号機 := 信号機 ++ {r |-> <進行>})
111 pre r in set dom 連動. 進路 and
112     進路設定 (r) = <未設定> and
113     (forall r1 in set 連動. 進路 (r). 鎖錠進路 & 進路設定 (r1) <> <設定>) and
114     (forall p in set dom 連動. 進路 (r). 転てつ器 &
115         転てつ器 (p) = 連動. 進路 (r). 転てつ器 (p) or
116         転てつ器の鎖錠 (p) = <解放>) and
117     (forall tc in set 連動. 進路 (r). 信号制御 & 軌道回路 (tc) = <非在線>)
118 post 進路設定 = 進路設定~ ++ {r |-> <設定>} ++
119                 {r1 |-> <鎖錠> | r1 in set 連動. 進路 (r). 鎖錠進路} and
120      (let 転換する転てつ器 =
121         {p | p in set dom 連動. 進路 (r). 転てつ器 & 転てつ器の鎖錠~(p) = <解放>} in
122         転てつ器の鎖錠 = 転てつ器の鎖錠~ ++
123            {p |-> <鎖錠> | p in set 転換する転てつ器} and
124         転てつ器 = 転てつ器~ ++
125                    {p |-> 連動. 進路 (r). 転てつ器 (p) | p in set 転換する転てつ器}) and
126      信号機 = 信号機~ ++ {r |-> <進行>};
127
128 列車移動 l (tc: 軌道回路名)
129 ext rd 連動 : 連動図表
130     rd 転てつ器 : map 転てつ器名 to 転てつ器の向き
131     rd 進路設定 : map 進路名 to 進路の状態
```

```
132      wr 信号機 : map 進路名 to 信号機の状態
133      wr 軌道回路 : map 軌道回路名 to 軌道回路の状態
134      wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
135  pre tc in set dom 軌道回路 and
136      軌道回路(tc) = <非在線> and
137      exists r in set dom 進路設定 &
138         進路設定(r) = <設定> and
139          tc in set 連動.進路(r).信号制御
140  post 信号機 = 信号機~ ++ {r |-> <停止> | r in set dom 進路設定 &
141              進路設定(r) = <設定> and tc in set 連動.進路(r).信号制御} and
142       軌道回路 = 軌道回路~ ++ {tc |-> <在線>} and
143       転てつ器の鎖錠 = 転てつ器の鎖錠 ++ {p |-> <鎖錠> |
144       p in set dom 転てつ器の鎖錠 & tc in set 連動.転てつ器(p)};
145
146  列車移動1 1 : 軌道回路名 ==> ()
147  列車移動1 1(tc) ==
148       (軌道回路 := 軌道回路 ++ {tc |-> <在線>};
149        信号機 := 信号機 ++ {r |-> <停止> | r in set dom 進路設定 &
150            進路設定(r) = <設定> and tc in set 連動.進路(r).信号制御};
151        転てつ器の鎖錠 := 転てつ器の鎖錠 ++ {p |-> <鎖錠> |
152            p in set dom 転てつ器の鎖錠 & tc in set 連動.転てつ器(p)})
153  pre tc in set dom 軌道回路 and
154      軌道回路(tc) = <非在線> and
155      exists r in set dom 進路設定 &
156         進路設定(r) = <設定> and
157          tc in set 連動.進路(r).信号制御;
158
159  列車移動2(tc : 軌道回路名)
160  ext rd 連動 : 連動図表
161      wr 軌道回路 : map 軌道回路名 to 軌道回路の状態
162      rd 信号機 : map 進路名 to 信号機の状態
163      rd 転てつ器 : map 転てつ器名 to 転てつ器の向き
164      wr 進路設定 : map 進路名 to 進路の状態
165      wr 転てつ器の鎖錠 : map 転てつ器名 to 鎖錠
166  pre tc in set dom 軌道回路 and
167      軌道回路(tc) = <在線>
168
169  post  軌道回路 = 軌道回路~ ++ {tc |-> <非在線>} and
170        (forall r in set dom 進路設定~ &
171           (進路設定~(r) = <未設定> => 進路設定~(r) = 進路設定(r)) and
172           (信号機(r) = <進行> => 進路設定(r) = 進路設定~(r)) and
173           ((信号機(r) = <停止> and 進路設定~(r) = <設定> and
174             forall tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <非在線>)
175                  => 進路設定(r) = <未設定>) and
176           ((信号機(r) = <停止> and 進路設定~(r) = <鎖錠中>) =>
177            進路設定(r) =
178                    if exists r1 in set dom 進路設定 &
179                       進路設定(r1) = <設定> and r1 in set 連動.進路(r).進路鎖錠
```

```
180                then <鎖錠中> else <未設定>)) and
181        (forall p in set dom 転てつ器の鎖錠 &
182           if (forall tc in set 連動.転てつ器(p) & 軌道回路(tc)=<非在線>) and
183            (forall r in set dom 進路設定 &
184                   進路設定(r) <> <設定> or p not in set dom 連動.進路(r).転てつ器)
185             then 転てつ器の鎖錠(p) = <解放>
186         else 転てつ器の鎖錠(p) = <鎖錠>);
187
188  列車移動２１ : 軌道回路名 ==> ()
189  列車移動２１(tc) ==
190     (軌道回路 := 軌道回路 ++ {tc |-> <非在線>};
191      for all r in set dom 進路設定 do
192      if 信号機(r) = <停止> and 進路設定(r) = <設定> and
193         forall tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <非在線>
194      then 進路設定 := 進路設定 ++ {r |-> <未設定>};
195      for all r in set dom 進路設定 do
196      if  信号機(r) = <停止> and 進路設定(r) = <鎖錠中>
197      then if not exists r1 in set dom 進路設定 & 進路設定(r1) = <設定> and
198       r1 in set 連動.進路(r).進路鎖錠
199            then 進路設定 := 進路設定 ++ {r |-> <未設定>};
200      for all p in set dom 転てつ器の鎖錠 do
201             if (forall tc in set 連動.転てつ器(p) & 軌道回路(tc)=<非在線>) and
202            (forall r in set dom 進路設定 &
203                     進路設定(r) <> <設定> or
204                 p not in set dom 連動.進路(r).転てつ器)
205             then 転てつ器の鎖錠 := 転てつ器の鎖錠 ++ {p |-> <解放>}
206     )
207  pre tc in set dom 軌道回路 and
208      軌道回路(tc) = <在線>
209  post  軌道回路 = 軌道回路~ ++ {tc |-> <非在線>} and
210        (forall r in set dom 進路設定~ &
211            (進路設定~(r) = <未設定> => 進路設定~(r) = 進路設定(r)) and
212            (信号機(r) = <進行> => 進路設定(r) = 進路設定~(r)) and
213            ((信号機(r) = <停止> and 進路設定~(r) = <設定> and
214                forall tc in set 連動.進路(r).進路鎖錠 & 軌道回路(tc) = <非在線>)
215                  => 進路設定(r) = <未設定>) and
216            ((信号機(r) = <停止> and 進路設定~(r) = <鎖錠中>) =>
217                進路設定(r) =
218                    if exists r1 in set dom 進路設定 &
219                 進路設定(r1) = <設定> and r1 in set 連動.進路(r).進路鎖錠
220                then <鎖錠中> else <未設定>)) and
221         (forall p in set dom 転てつ器の鎖錠 &
222             if (forall tc in set 連動.転てつ器(p) & 軌道回路(tc)=<非在線>) and
223            (forall r in set dom 進路設定 &
224                     進路設定(r) <> <設定> or p not in set dom 連動.進路(r).転てつ器)
225             then 転てつ器の鎖錠(p) = <解放>
226         else 転てつ器の鎖錠(p) = <鎖錠>);
227
```

```
values
R1 = mk_token("1R");
R2 = mk_token("3R");
R3 = mk_token("2L");
R4 = mk_token("4L");
R5 = mk_token("1L");

TC1 = mk_token("1RT");
TC2 = mk_token("4LT");
TC3 = mk_token("11T");
TC4 = mk_token("21T");
P1 = mk_token("11");
P2 = mk_token("22");

RR1 = mk_進路条件 ({P1 |-> <定位>}, {}, {TC3, TC1}, {TC3});
RR2 = mk_進路条件 ({P2 |-> <反位>}, {}, {TC4}, {TC4});
RR3 = mk_進路条件 ({P2 |-> <定位>}, {}, {TC4, TC2}, {TC4});
RR4 = mk_進路条件 ({P1 |-> <反位>}, {R5}, {TC3}, {TC3});
RR5 = mk_進路条件 ({P1 |-> <反位>}, {R4}, {TC3, TC2}, {TC3});

IL = mk_連動図表 ({R1 |-> RR1, R2 |-> RR2, R3 |-> RR3, R4 |-> RR4, R5 |-> RR5},
   {P1 |-> {TC3}, P2 |-> {TC4}}, {TC1, TC2, TC3, TC4})
```

付録 B

線区データベース仕様（VDM）

B.1　非形式的仕様

本章ではディジタル ATC データベースの具体的な非形式的仕様を載せる．

B.1.1　基本構造 レベル 1（エリアに関する定義）

概要

1. エリアとは軌道回路および進路の集合である．

構成要素

2. エリアには以下の要素がある．

（1）軌道回路の集合 （3., 9. 参照）
（2）軌道回路内進路の集合 （10., 11. 参照）
（3）進路の集合（13., 14. 参照）
（4）単純な（Plain）エリアか複雑な（Complex）エリアかの区別
（5）エリア内の最高速度（km/h）

3. **軌道回路**

概要

（1）軌道回路とは軌道を構成する基本単位である．軌道は一定の区間に分割される．それぞれの区間には，その上に列車が存在するかどうかを検知できる軌道回路と呼ばれる装置が設置される．転じてその単位そのものを軌道回路と呼ぶ．軌道回路には列車を 2 方向に振り分ける分岐が含まれることもある．

要素

（2）軌道回路には以下の要素がある．

a. 境界の集合（4. 参照）
b. ATC 信号の使用状況および搬送波（6. 参照）（A/B のそれぞれの方向（5. 参照）に関して登録される）
c. TD 信号の使用状況および搬送波（7. 参照）
d. AT/BT の別（8. 参照）

要求事項

（3）1 軌道回路に含まれる境界は 2 つ以上であること．
（4）軌道回路が無絶縁軌道回路の場合，TD 波が使用されていること．
（5）もし ATC が両方向で使用されている場合，ATC 搬送波は A 方向と B 方向で異なること．

4.（軌道回路）境界

概要

（1）複数の軌道回路が互いに接続しているが，この接続している場所を軌道回路境界と呼ぶ（以後，境界と呼ぶ）.

（2）境界には固有の ID をつける．この境界は軌道回路内で一意である．

（3）境界はその電気的特性で 2 種類に分けられる．無絶縁境界と有絶縁境界である．有絶縁境界の場合，隣会う 2 つの軌道回路は電気的に独立している．無絶縁境界の場合，それぞれの電流が他の軌道回路に流れるため，異なる搬送波を使用する必要がある．

要素

（4）境界には以下の要素がある．

a. 位置（起点からの距離を m 単位で示す）

b. 絶縁の種類 – 有絶縁/無絶縁

c. ATC 区間の始端あるいは終端であるか（始端/終端の区別は不用）．A 方向，B 方向のそれぞれについて登録する．

d. 線路の終端であるか．すなわち，他の軌道回路と接続しないかどうか．仮に終端である場合，この境界（実際には物理的な接続はない）は 1 つの軌道回路にのみ属する．

5. 方向

方向を軌道回路に設定する．距離（キロ程）が増大する方向を A 方向を，減少する方向を B 方向と定める．

6. ATC 信号

ATC 信号は軌道回路に使用される信号の 1 つで軌道回路を用いてその軌道回路に関する情報を列車に送るのに使用される．複数の搬送波の使用が想定される．もし，この軌道回路が ATC による制御区間の外にある場合には ATC 信号は使用されない．この場合，従来 ATC である場合もあるし，その他のシステムが使用されていることもある．

7. TD signal

軌道回路における列車検知は有絶縁軌道回路（すべての境界が絶縁されている）では ATC 信号を使うが（ただし，ATC を使用していないところではその限りではない），無絶縁軌道回路（絶縁されていない境界がある場合）では，それでは不十分で，TD（Train Detection: 列車検知）波が使用される．この場合，隣接軌道回路では，異なる搬送波が使用される[*1]．

8. AT/BT – TD 波の送信方法

TD 信号を軌道回路の前方（終点方）から送るのを AT，後方（起点方）から送るのを BT と呼ぶ.

9. 軌道回路の集合

概要

（1）軌道回路毎に ID を付与する．この ID はエリアの中で一意とする．ID と軌道回路の情報の対をエリアに登録する．

要求事項

（2）同じ境界を所有するのは 2 つまでの軌道回路である．もし 1 つの場合は，それは線路の終端を意味する．

（3）もし境界が 2 つの軌道に属している場合，その隣接軌道回路間で以下に示す整合性が求められる．その整合性とは

a. それぞれの軌道回路に登録されている境界の情報は同一である．

[*1] 実際には送信と受信が互い違いになるようにし，送信箇所では 2 軌道回路分を同一信号を用いて送信，受信側では異なる 2 軌道回路分の信号を受信する．

b. その境界が線路の終端となっていないこと．
c. ATC 搬送波は隣接軌道回路で異なっていること[*2]．
d. もし境界が無絶縁である場合，TD 波は異なること[*3]．

10. **軌道回路内進路（Path）**

概要

（1）軌道回路内進路（以後，Path と呼ぶ）は列車が軌道回路をどのように進むかを示すものである．境界の組として定義される．

要素

（2）Path は以下の情報を持つ．
a. 軌道回路 ID. この ID で示される軌道回路が当該の Path を含むことが要求される．
b. 始端となる境界 ID
c. 終端となる境界 ID
d. 長さ（m）– 始端と終端の間の距離
e. A 方向について使用されるか
f. B 方向について使用されるか
g. 各種制限 （12. 参照）

要求事項

（3）始端と終端は異なる境界である．
（4）終端の位置は始端の位置より大きい．
（5）長さは終端と始端の位置の差に等しい．
（6）A 方向あるいは B 方向の少なくともいずれかは使用されていること．
（7）各種制限に関する条件．（12. 参照）．

11. **軌道回路内進路（Path）の集合**

概要

（1）それぞれの Path に ID を付与する．この ID はエリア内（軌道回路内ではない）で一意である．

要求事項

（2）異なる ID を持つ 2 つの軌道回路内進路が同一の始端と終端を持つことは認めない．
（3）同一の境界が同じ軌道回路に属する軌道回路内進路の始端と終端に同時になることは認めない．

12. **各種制限**

概要

（1）速度制限を軌道回路内進路に設定できる．列車の一部でもこの区間にかかっていれば，列車はその速度以下で走行する必要がある．
（2）勾配の大きさを軌道回路内進路に登録できる．
（3）き電区分制御軌道回路，あるいはセクションの位置を軌道回路内進路に登録できる．これはき電（列車への送電）に関連した装置である[*4]．
（4）速度制限，勾配，セクションを各種制限として扱う．もしこの各種制限を軌道回路を跨って設定したい場合は，それぞれの軌道回路内進路に設定する．

[*2] これは，絶縁が故障しても列車検知を安全に行うためだが，JR 東日本の DS-ATC，JR 東海の ATC-NS とも軌道回路 ID に整合性がない場合には異常を検知できることから同一周波数信号を用いている．

[*3] 同一周波数であっても起点方が AT，終点方 BT であれば問題ないのが正当で，通常は起点方 AT，終点方 BT の場合は同一周波数を用いる．また，通常は AT と BT を交互に用いる．

[*4] 交流電化区間では 2 つの変電所からの送信電力の位相が異なることがありうる．それらが短絡してしまうと大電流が流れてしまうため，セクションを設置し，区分できるようにしている．新幹線では両方の変電所から区分される中セクションを設け，この中セクション付近にき電区分制御軌道回路を設置し，この区間に列車がある時に，この区間へ電力を供給する変電所を切り替える．在来線（海外の高速鉄道もそうであるが）では電力を供給しない区間を設ける．従ってこの場合はここで列車は停止できない．

要素

（5）各種制限は以下の要素からなる．

a. 種別 – 速度制限，勾配，セクション
b. 開始位置
c. 終了位置
d. 速度制限はその最高速度（km/h）
e. 勾配についてはその大きさ．単位はパーミルで 0.1 パーミルで設定可能とする．A 方向に向かって走行している時に上り勾配になるときに正，下り勾配になる時に負とする．
f. セクションについては，特にその他の情報はないものとする．

要求事項

（6）開始位置と終了位置は軌道回路内進路の始端と終端の間（両端を含む）にあること．

（7）速度制限は重なることを認める．（実際には最も速度が小さいものを採用する）

（8）勾配は重なることを認めない．

（9）セクションは重なることを認めない．

13. 進路

概要

（1）進路は Path の列として定義される．ディジタル ATC 制御区間では，列車は進路のみを走行することが認められる．

要素

（2）進路には以下の要素がある．

a. 進路の方向（A/B）
b. Path の列

14. 進路の集合

概要

（1）それぞれの進路に ID が付与される．この ID はエリア内で一意である．ID と進路の情報の対がエリアに登録される．

エリアに対する要求事項

軌道回路の集合，Path の集合および進路の集合の関係に関する要求事項は以下のとおりである．

15. 軌道回路内進路の軌道回路 ID に示された軌道回路が登録されており，始端，終端の境界も存在する．

16. 進路に登録されている Path が登録されており，進路に定義された方向に関して使用可能である．

17. 進路に登録されている軌道回路内進路はその順番に接続されている．

18. 進路は循環しない．すなわち，列の中に同一の軌道回路を含まない．

19. 進路に登録されている軌道回路内進路に関し，ATC 信号が使用可能である．（ATC が使用可能でない場合は進路を設定しない）

エリアを操作するための関数

20. **Add-TrackC**

機能

（1）新たに軌道回路をエリアに登録する．

入力

（2）この関数では以下の値を入力する

a. 編集するエリア
b. 新たに登録する軌道回路に付与する ID
c. 新しい軌道回路の情報（境界，ATC 信号，TD 信号，AT/BT の別）

要求事項

（3）指定した軌道回路 ID がすでにエリアに登録されている軌道回路 ID と一致しないこと．

（4）新しい軌道回路に含まれる境界がすでに存在する場合には，既存軌道回路でその境界を登録しているものは高々 1 つであること．

（5）新しい軌道回路と隣合う（接続する）既存軌道回路とは，境界および信号について 9（3）で示される整合性を持つこと．

21. **Del-TrackC**

機能

（1）軌道回路をエリアから削除する．

入力

（2）この関数では以下の値を入力する．

a. 編集するエリア

b. 削除する軌道回路の ID

要求事項

（3）指定された軌道回路 ID がエリアに登録されていること．

（4）削除しようとする軌道回路に対して，それに属する Path が存在しないこと．

22. **Add-Joint**

機能

（1）新たに軌道回路に境界を追加登録する．

入力

（2）この関数では以下の値を入力する．

a. 編集するエリア

b. 編集する軌道回路の ID

c. 新たに登録する境界に付与する ID

d. 新しい境界の情報（位置，絶縁等）

要求事項

（3）指定した軌道回路 ID がエリアに登録されていること．

（4）指定された ID に対応する軌道回路が有絶縁軌道回路の場合，新しい境界も有絶縁である．

（5）新しい境界の ID が別の軌道回路に登録されている場合，そのような既存軌道回路は高々 1 つである．

（6）編集する軌道回路と，追加する境界をすでに所有している既存軌道回路とは，境界および信号について 9（3）で示される整合性を持つこと．

23. **Del-Joint**

機能

（1）軌道回路から境界を削除する．境界が 2 つの軌道回路に所属している場合に両方の軌道回路から境界を 1 度に削除することはしないので，両方から削除する場合にはそれぞれの軌道回路について呼び出す必要がある．2 つ以下の場合は削除ができない．

入力

（2）この関数では以下の値を入力する．

a. 編集するエリア

b. 境界を削除したい軌道回路 ID

c. 削除したい境界の ID

要求事項

（3）指定された軌道回路 ID がエリアに登録されていること．

（4）指定された境界 ID が指定された軌道回路に存在すること．

（5）指定された軌道回路に境界が 3 つ以上あること．
（6）指定された軌道回路に属し，削除しようとする境界を有する Path が存在しないこと．

24. Add-Path

機能
（1）新たに Path を登録する．
入力
（2）この関数は以下の値を入力する．
a. 編集するエリア
b. 新たに登録する Path に付与する ID
c. 新しい Path の情報（始端，終端の境界，長さ，各方向の使用可否）
要求事項
（3）指定した Path ID がすでにエリアに登録されている Path ID と一致しないこと．
（4）始端，終端の境界がすでに軌道回路に登録されているものであること．
（5）始端の位置が終端の位置より小さいこと．

25. Del-Path

機能
（1）Path をエリアから削除する．
入力
（2）この関数では以下の値を入力する．
a. 編集するエリア
b. 削除しようとする Path ID
要求事項
（3）指定した Path ID がエリアに登録されていること．
（4）削除しようとする Path を有する進路が存在しないこと．

26. Add-Route

機能
（1）新たに進路をエリアに登録する．
入力
（2）この関数は以下の入力を必要とする．
a. 編集するエリア
b. 登録する進路に付与する ID
c. 登録する進路の情報，すなわち進路の方向と Path の列
要求事項
（3）指定した進路 ID がすでにエリアに登録されている進路 ID と一致しないこと．
（4）登録する進路に含まれる Path が全て登録済みであり，進路の方向に関して使用可能であること．
（5）登録する進路に含まれる Path は列に示される順番で接続できること．
（6）登録する進路に含まれる Path に対し，進路の方向に関して ATC 信号が設備されていること．

27. Del-Route

機能
（1）進路をエリアから削除する．
入力
（2）この関数では以下の値を入力する．
a. 編集するエリア
b. 削除しようとする進路の ID
要求事項

（3）指定した進路 ID がエリアに登録されていること．

28. **Add-Condition**

機能

（1）制限を Path に設定する．エリアに対して提供される．ただし，既存の制限と新規の制限をつなげて新しい制限として登録し直すことはしないものとする[*5]．

入力

（2）編集するエリア

（3）制限を設定する Path の ID

（4）制限の情報（種別，位置，速度，勾配等）

要求事項

（5）指定した Path ID がエリアに登録されていること．

（6）制限が Path の始端と終端の間に存在すること．

（7）もし，制限が勾配であるならば，すでに存在する勾配と重ならないこと．

（8）もし，制限がセクションであるならば，すでに存在するセクションと重ならないこと．

29. **Del-Condition**

機能

（1）制限を Path から外す．エリアに対して提供される．ただし，設定された制限の一部だけを外して，新しい制限として登録する機能は提供しない[*6]．

入力

（2）編集するエリア

（3）制限を外す Path の ID

（4）制限の種別

（5）位置

要求事項

（6）指定した PathID が存在すること．

（7）種別，開始，終了位置が一致する制限が存在すること．

B.1.2 基本構造 レベル 2（線区に関する定義）

概要

1. 線区は本来はエリアの列で定義される．しかし，分岐等も考えうるため，エリアの集合として定義する．

要素

2. 線区には以下の要素が含まれる．

（1）エリアの集合（3 および レベル 1. 参照）

（2）エリア間接続の集合（4 参照）

3. エリアの集合

概要

（1）エリア毎に ID を付与する．この ID は線区内で一意である．エリア ID とエリアの情報の対を線区に登録する．

[*5] 複雑になるため．機能としては必要になるかもしれない．

[*6] 複雑になるため

4. エリア間接続

概要

（1）エリアは互いに接続ができる．この接続は異なるエリアに属する境界の対で定義される．

（2）この接続に登録される境界は物理的には同一のものであるが，それぞれのエリアで情報を持っている．ID についても同じである必要はない．

要素

（3）エリアに属する境界の対

（4）それぞれの境界はエリア ID，軌道回路 ID，境界 ID で指定される．

（5）方向がこの接続で変わるかどうかの情報．これが真であれば，方向の変更（この接続点で異なったエリアに属する 2 つのエリアの軌道回路内進路の終端同士，あるいは始端同士が接続することができる）が許される．

（6）距離がこの接続で変わるか．これが真であれば，距離の変更（2 つのエリアで異なる距離を持つ）が許される．

要求事項

（7）対となっている境界はエリアが異なる必要がある．

（8）エリア間接続に登録されている境界は，それぞれのエリアでは 1 つの軌道回路にのみ属している．もしいずれかのエリアで 2 つの軌道回路に属している場合には接続はできない．

（9）同じ境界を同時に 2 つ以上のエリア間接続に登録することはできない．

要求事項

5. エリア接続に登録された境界はそれぞれのエリアに存在する．

6. エリア接続に登録された境界のそれぞれのエリアでの情報について，レベル 1. 9（3）で示される境界の整合性がとれている．ただし，以下を例外とする．

（1）方向が変更されると宣言された場合には，方向が変更される．

（2）距離が変更されると宣言された場合には，距離が変更される．

エリアを操作するための関数

7. **Add-Area**

機能

（1）新たに空のエリアを線区に登録する．エリア内の要素は別途追加していく．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 新たに登録するエリアに付与する ID

要求事項

（3）指定したエリアがすでに線区に登録されているエリア ID と一致しないこと．

8. **Change-Area**

機能

（1）エリアの中身を新しいものに変更する．この操作は他の操作から呼び出されることを前提とする．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 変更するエリアの ID

c. 新しいエリアの情報

要求事項

（3）指定したエリア ID が線区に登録されていること．

（4）線区に関する要求事項が保持されること．

9. Del-Area

機能

（1）エリアを線区から削除する．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 削除しようとするエリアの ID

要求事項

（3）指定したエリア ID が線区に登録されていること．

（4）削除しようとするエリアに関連したエリア間接続が存在しないこと．

10. Add-Connect

機能

（1）新たにエリア間接続を線区に登録する．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 接続しようとする境界の対（エリア ID，軌道回路 ID，境界 ID を指定する）

c. 方向がこの接続箇所で変わるかどうか．

d. 距離がこの接続箇所で変わるかどうか．

要求事項

（3）指定した境界がエリア間接続として登録されていないこと．

（4）指定した境界が存在していること．ただし，1 つの軌道回路にのみ属していること．

（5）指定した境界の対が整合性を持っており，隣接する軌道回路の条件も整合性を持っていること．具体的には 6. に示す条件を満たす必要がある．

11. Del-Connect

機能

（1）エリア間接続を線区から削除する．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 削除しようとする境界のうちの一方（もう一方も削除される）

要求事項

（3）指定した境界がエリア間接続に登録されていること．

12. Line-Add-TrackC

機能

（1）新たに軌道回路を線区に登録する．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 軌道回路を登録するエリアの ID

c. 新たに登録する軌道回路に付与する ID

d. 新しい軌道回路の情報（境界，ATC 信号，TD 信号，AT/BT の別）

要求事項

（3）エリア ID が線区に登録されているこ．と

（4）エリア ID で指定されたエリアに関して，レベル 1.20 （Add-TrackC）の要求事項が満たされていること．
（5）新しい軌道回路の全ての境界について，エリア間接続に登録されていないこと

13. Line-Del-TrackC

機能

（1）軌道回路を線区から削除する．

入力

（2）この関数では以下の値を入力する．
a. 編集する線区
b. 削除する軌道回路が登録されているエリアの ID
c. 削除する軌道回路の ID

要求事項

（3）エリア ID が線区に登録されていること．
（4）エリア ID で指定されたエリアに関して，レベル 1.21（Del-TrackC）の要求事項が満たされていること．
（5）削除しようとする軌道回路に含まれる境界が，エリア間接続に登録されていないこと．

14. Line-Add-Joint

機能

（1）新たに線区内の軌道回路に境界を追加登録する．

入力

（2）この関数では以下の値を入力する．
a. 編集する線区
b. 新たに境界を追加するエリアの ID
c. 新たに境界を追加する軌道回路の ID
d. 新たに登録する境界に付与する ID
e. 新しい境界の情報（位置，絶縁等）

要求事項

（3）指定したエリア ID が線区に登録されていること．
（4）指定した軌道回路 ID が指定されたエリアに登録されていること．
（5）エリア ID で指定されたエリアに関して，レベル 1．22 （Add-Joint）の要求事項が満たされていること．

15. Line-Del-Joint

機能

（1）線区から境界を削除する．境界が 2 つの軌道回路に所属している場合に両方の軌道回路から境界を 1 度に削除することはしないので，両方から削除する場合にはそれぞれの軌道回路について呼び出す必要がある．

入力

（2）この関数では以下の値を入力する．
a. 編集する線区
b. 削除する境界が登録されているエリアの ID
c. 境界を削除したい軌道回路 ID
d. 削除したい境界の ID

要求事項

（3）指定したエリア ID が線区に登録されていること．
（4）指定した軌道回路 ID が指定されたエリアに登録されていること．

（5）指定された境界 ID が指定された軌道回路に存在すること．
（6）エリア ID で指定されたエリアに関して，レベル 1.23 （Del-Joint）の要求事項が満たされていること．
（7）削除しようとする境界がエリア間接続に登録されていないこと．

16. Line-Add-Path

機能

（1）新たに線区に Path を登録する．

入力

（2）この関数は以下の値を入力する．
a. 編集する線区
b. 新たに Path を追加するエリアの ID
c. 新たに登録する Path に付与する ID
d. 新しい Path の情報（始端，終端の境界，長さ，各方向の使用可否）

要求事項

（3）指定したエリア ID が線区に登録されていること．
（4）エリア ID で指定されたエリアに関して，レベル 1.24 （Add-Path）の要求事項が満たされていること．
（5）追加する Path の境界がエリア間接続に登録されている場合は，接続されている他のエリアの Path と情報の整合性を持っていること．

17. Line-Del-Path

機能

（1）Path を線区から削除する．

入力

（2）この関数では以下の値を入力する．
a. 編集する線区
b. 削除しようとする Path が登録されているエリアの ID
c. 削除しようとする Path ID

要求事項

（3）指定した Path ID がエリアに登録されていること．
（4）エリア ID で指定されたエリアに関して，レベル 1.25 （Del-Path）の要求事項が満たされていること．

18. Line-Add-Route

機能

（1）新たに進路を線区に登録する．

入力

（2）この関数は以下の入力を必要とする．
a. 編集する線区
b. 新たに進路を追加するエリアの ID
c. 登録する進路に付与する ID
d. 登録する進路の情報，すなわち進路の方向と Path の列

要求事項

（3）指定したエリア ID が線区に登録されていること．
（4）エリア ID で指定されたエリアに関して，レベル 1.26 （Add-Route）の要求事項が満たされていること．

19. **Line-Del-Route**

機能

（1）進路を線区から削除する．

入力

（2）この関数では以下の値を入力する．

a. 編集する線区

b. 削除しようとする進路が登録されているエリアの ID

c. 削除しようとする進路の ID

要求事項

（3）指定したエリア ID が線区に登録されていること．

（4）エリア ID で指定されたエリアに関して，レベル 1.27 （Del-Route）の要求事項が満たされていること．

20. **Line-Add-Condition**

機能

（1）制限を Path に設定する．線区に対して提供される．

入力

（2）編集する線区

（3）制限を設定するエリアの ID

（4）制限を設定する Path の ID

（5）制限の情報（種別，位置，速度，勾配等）

要求事項

（6）編集するエリアの ID が存在すること．

（7）エリア ID で指定されたエリアに関して，レベル 1.28 （Add-Condition）の要求事項が満たされていること．

21. **Line-Del-Condition**

機能

（1）制限を Path から外す．線区に対して提供される．

入力

（2）編集する線区

（3）編集するエリアの ID

（4）制限を外す軌道回路内進路の ID

（5）制限の種別

（6）位置

要求事項

（7）編集するエリアの ID が存在すること．

（8）エリア ID で指定されたエリアに関して，レベル 1.29 （Del-Condition）の要求事項が満たされていること．

B.1.3　完成した（運用に使用できる）データベース構造の条件

これまでに記述した条件は常に成り立っている必要があるが，実際に運用するデータベースにはさらに条件が必要である．

1. 任意の軌道回路境界は軌道の終端であるか，他のエリアへの接続となっている．

2. 任意の軌道回路境界に対して，同一軌道回路に属する他の軌道回路境界との Path が存在する．

3. 任意の軌道回路のいずれの方向においても，ATC 信号が使用できる場合には，その方向に対する Path が使用可能である．

4. 任意の Path に対して，ATC 信号が存在する場合には，その Path を含む進路が必要である．
5. 利用可能な方向に関して，任意の Path は 1 つ以上の他の Path から接続されるか（エリアが異なっていてもよい），ATC 制御区間始端であるか，軌道始端である．
6. 利用可能な方向に関して，任意の Path は 1 つ以上の他の Path に接続しているか（エリアが異なっていてもよい），ATC 制御区間終端であるか，軌道終端である．
7. 任意の進路に対して，その後に続く進路が存在し，それをたどることにより，他のエリアへの接続，ATC 制御区間終端あるいは軌道終端にたどりつくことができる．

B.2 形式仕様

B.2.1 基本構造 レベル 1 – エリアに関する定義

構成要素および不変条件

```
--
-- Model of Digital ATC Database
--
-- ver 4.61 2001.2.5 by n.terada
-- track circuits -- basic unit
--
types
TrackC:: joints : map Joint_id to Joint
         atc : map Direction to ATC
         td : TD
         atbt : ATBT
inv tc == card dom tc.joints > 1 and
          dom tc.atc = {<ADIR>, <BDIR>} and
          TD_Used_for_NonInsulated_TrackC(tc.td, tc.atbt, rng tc.joints) and
          (tc.atc(<ADIR>).used and tc.atc(<BDIR>).used =>
               tc.atc(<ADIR>).carrier <> tc.atc(<BDIR>).carrier);

--
-- joint -- connection between two track circuits
--

Joint_id = token;

Joint:: position : nat
        insulated : Insulation
        remark : Remark;

Insulation = bool; -- true if joint is insulated

Remark :: atc_terminal : map Direction to bool  -- true if atc is terminated
          line_terminal : bool -- true if line terminated
inv rm == dom rm.atc_terminal = {<ADIR>, <BDIR>};

Direction = <ADIR> | <BDIR>;

--
-- carrier of track circuits
-- atc signal , td(train detection signal)
--

ATC :: used : bool -- true if Digital ATC is used
       carrier : token;

TD :: used : bool  -- true if TD is used
      carrier : token;

ATBT = <AT> | <BT> | <NULL>;

functions
TD_Used_for_NonInsulated_TrackC : TD * ATBT * set of Joint -> bool
TD_Used_for_NonInsulated_TrackC(td, atbt, joints) ==
    (atbt = <NULL> <=> not td.used) and
    ((exists j in set joints & not j.insulated) => td.used);

--
-- Collection of Track Circuits
```

```
--
types
TrackC_id = token;
TrackC_map = map TrackC_id to TrackC
inv tcs == forall tid in set dom tcs &
           forall jid in set dom tcs(tid).joints &
               Only_One_Next_TrackC(tcs, tid, jid) and
               forall tid2 in set dom tcs & tid <> tid2 =>
                           Joint_and_Next_TrackC(tcs(tid), tcs(tid2), jid);

functions
Only_One_Next_TrackC: map TrackC_id to TrackC * TrackC_id * Joint_id -> bool
Only_One_Next_TrackC(tcs, tcid, jid) ==
    card {tid | tid in set dom tcs &
                tid <> tcid and jid in set dom tcs(tid).joints} < 2;

Joint_and_Next_TrackC: TrackC * TrackC * Joint_id -> bool
Joint_and_Next_TrackC(tc1, tc2, jid) ==
    (jid in set dom tc1.joints and jid in set dom tc2.joints) =>
        (tc1.joints(jid) = tc2.joints(jid) and
         not tc1.joints(jid).remark.line_terminal and
         Is_wf_adjacent_signal(tc1, jid, tc2, jid, false));

--
-- condition related with signal
--
Is_wf_adjacent_signal: TrackC * Joint_id * TrackC * Joint_id * bool -> bool
Is_wf_adjacent_signal(tc1, jid1, tc2, jid2, dir_chng) ==
    jid1 in set dom tc1.joints and
    jid2 in set dom tc2.joints and
    Remark_Compatible(tc1.joints(jid1).remark, tc2.joints(jid2).remark, dir_chng) and
    ATC_Terminal_and_ATC_Used(tc1.atc(<ADIR>), tc2.atc(<BDIR>),
         tc1.joints(jid1).remark, tc2.atc(<ADIR>), tc2.atc(<BDIR>),
         tc2.joints(jid2).remark, dir_chng) and
    Adjacent_TD_Carrier_Differ(tc1.td, tc1.atbt, tc2.td, tc2.atbt,
                tc1.joints(jid1).insulated and tc2.joints(jid2).insulated);

Remark_Compatible : Remark * Remark * bool-> bool
Remark_Compatible(rm1, rm2, dir_chng) ==
    (not dir_chng => rm1.atc_terminal = rm2.atc_terminal) and
    (dir_chng =>
        (rm1.atc_terminal(<ADIR>) = rm2.atc_terminal(<BDIR>) and
         rm1.atc_terminal(<BDIR>) = rm2.atc_terminal(<ADIR>)));

ATC_Terminal_and_ATC_Used : ATC * ATC * Remark * ATC * ATC * Remark * bool -> bool
ATC_Terminal_and_ATC_Used(atcA1, atcB1, rm1, atcA2, atcB2, rm2, dir_chng) ==
    (not dir_chng =>
        (((atcA1.used <> atcA2.used) = rm1.atc_terminal(<ADIR>)) and
         ((atcB1.used <> atcB2.used) = rm1.atc_terminal(<BDIR>)))) and
    (dir_chng =>
        (((atcA1.used <> atcB2.used) = rm1.atc_terminal(<ADIR>)) and
         ((atcB1.used <> atcA2.used) = rm1.atc_terminal(<BDIR>))))

pre     Remark_Compatible(rm1, rm2, dir_chng);

Adjacent_TD_Carrier_Differ : TD * ATBT * TD * ATBT * Insulation -> bool
Adjacent_TD_Carrier_Differ(td1, atbt1, td2, atbt2, insulated) ==
    (insulated or
        td1.carrier <> td2.carrier and
        (atbt1 <> atbt2 or atbt1 = <AT> and atbt2 = <AT>));
--
-- end of track circuit
--

--
-- paths in track circuits
--
types
Path:: tc : TrackC_id
       start : Joint_id
       endp : Joint_id
       length : nat
       used : map Direction to bool
       condition : set of Condition
inv p == p.start <> p.endp and
         dom p.used = {<ADIR>, <BDIR>} and
         (exists dr in set dom p.used & p.used(dr)) and
```

```
134           (forall c1, c2 in set p.condition & Condition_not_Conflict(c1, c2));
135   -- start.position < endp.position is added at inv_Area
136
137   Condition :: kind : Cond_Kind
138                start : nat
139                endp : nat
140                speed : nat     -- only used for <LIMIT>
141                permill : nat   -- should be integer or real
142   inv con == con.start < con.endp;
143
144   Cond_Kind = <LIMIT> | <GRADIENT> | <SECTION>;
145
146   functions
147   Condition_not_Conflict : Condition * Condition -> bool
148   Condition_not_Conflict(c1, c2) ==
149       (c1 <> c2 and c1.kind = c2.kind and c1.kind <> <LIMIT>) =>
150            not Overlap(c1, c2);
151
152   Overlap : Condition * Condition -> bool
153   Overlap(c1, c2) ==
154       (c1.start < c2.start and c2.start < c1.endp) or
155       (c2.start < c1.start and c1.start < c2.endp) or
156       (c1.start = c2.start);
157
158   types
159   Path_id = token;
160
161   Path_map = map Path_id to Path
162   inv ps == forall pid1, pid2 in set dom ps & pid1 <> pid2 =>
163                   (Not_Same_Path(ps(pid1), ps(pid2)) and
164                    Not_Start_and_End(ps(pid1), ps(pid2)));
165
166   functions
167   Not_Same_Path : Path * Path -> bool
168   Not_Same_Path(p1, p2) ==
169       not (p1.start = p2.start and p1.endp = p2.endp) and
170       not (p1.start = p2.endp and p1.endp = p2.start);
171
172   -- It is possible to allow the case p1.tc <> p2.tc
173   -- but that is not practical, so it is not allowed here.
174
175   Not_Start_and_End : Path * Path -> bool
176   Not_Start_and_End(p1, p2) ==
177       (p1.tc = p2.tc) =>
178           (not p1.start = p2.endp and not p2.start = p1.endp);
179
180   --
181   -- route
182   --
183   types
184   Route:: dr : Direction
185           paths : seq1 of Path_id;
186
187   Route_map = map Route_id to Route;
188
189   Route_id = token;
190
191   --
192   -- Area -- corresponds to stations or intermediate part.
193   --
194
195   Area :: trackcs : TrackC_map
196           paths : Path_map
197           routes : Route_map
198           kind : Area_Kind
199           max : MaxSpeed
200
201   inv mk_Area(trackcs, paths, routes, -, -) ==
202       (forall p in set rng paths &
203           Path_within_TrackC(trackcs, p) and
204           Direction_Correct(trackcs, p)) and
205       (forall r in set rng routes &
206           Path_Exists(paths, r.paths, r.dr) and
207           Exists_ATC_for_Route(trackcs, paths, r) and
208           Route_not_Circular(paths, r) and
209           Path_Connected(paths, r.paths, r.dr));
210
```

```
Area_Kind = <PLAIN> | <COMPLEX>;
MaxSpeed = token;

functions
Path_within_TrackC : TrackC_map * Path -> bool
Path_within_TrackC(trackcs, p) ==
    p.tc in set dom trackcs and
    p.start in set dom trackcs(p.tc).joints and
    p.endp in set dom trackcs(p.tc).joints;

Direction_Correct : TrackC_map * Path -> bool
Direction_Correct(trackcs, p) ==
    let pstart = trackcs(p.tc).joints(p.start).position,
        pend = trackcs(p.tc).joints(p.endp).position in
            pstart < pend and
            p.length = pend - pstart and
            forall c in set p.condition &
                  pstart <= c.start and c.endp <= pend
pre Path_within_TrackC(trackcs, p);

Path_Exists : Path_map * seq of Path_id * Direction -> bool
Path_Exists(paths, route, dr) ==
    forall pid in set elems route &
        pid in set dom paths and
        paths(pid).used(dr);

-- next function is related with signal
Exists_ATC_for_Route : TrackC_map * Path_map * Route -> bool
Exists_ATC_for_Route(trackcs, paths, r) ==
    forall pid in set elems r.paths &
        paths(pid).tc in set dom trackcs and
        trackcs(paths(pid).tc).atc(r.dr).used
pre       Path_Exists(paths, r.paths, r.dr);

Route_not_Circular : Path_map * Route-> bool
Route_not_Circular(paths, r) ==
    forall i, j in set inds r.paths &
        i <> j => paths(r.paths(i)).tc <> paths(r.paths(j)).tc
pre       Path_Exists(paths, r.paths, r.dr);

Path_Connected : Path_map * seq of Path_id * Direction-> bool
Path_Connected(paths, route, dr) ==
    forall i in set inds route &
        (i + 1) in set inds route =>
             ((dr = <ADIR> =>
                   paths(route(i)).endp = paths(route(i+1)).start) and
              (dr = <BDIR> =>
                   paths(route(i)).start = paths(route(i+1)).endp))
pre       Path_Exists(paths, route, dr);

--
-- end of invariant
--
```

エリアに関する操作

```
--
-- Operation of Data Base (Area Level)
--
Add_TrackC : Area * TrackC_id * TrackC -> Area
Add_TrackC(ar, tcid, tc) ==
    mu(ar, trackcs |-> ar.trackcs ++ {tcid |-> tc})
pre tcid not in set dom ar.trackcs and
    forall jid in set dom tc.joints &
        Only_One_Next_TrackC(ar.trackcs, tcid, jid) and
        forall tcid1 in set dom ar.trackcs &
            Joint_and_Next_TrackC(tc, ar.trackcs(tcid1), jid)
post tcid in set dom RESULT.trackcs and
     RESULT.trackcs = ar.trackcs ++ {tcid |-> tc} and
     RESULT.trackcs(tcid) = tc and
     RESULT.paths = ar.paths and
     RESULT.routes = ar.routes;

Del_TrackC : Area * TrackC_id -> Area
Del_TrackC(ar, tcid) ==
```

```
284       mu(ar, trackcs |-> {tcid} <-: ar.trackcs)
285   pre tcid in set dom ar.trackcs and
286       forall p in set rng ar.paths & p.tc <> tcid
287   post tcid not in set dom RESULT.trackcs and
288        RESULT.trackcs = {tcid} <-: ar.trackcs and
289        RESULT.paths = ar.paths and
290        RESULT.routes = ar.routes;
291
292   Add_Joint : Area * TrackC_id * Joint_id * Joint -> Area
293   Add_Joint(ar, tid, jid, joint) ==
294       let tc = ar.trackcs(tid) in
295           mu(ar, trackcs |-> ar.trackcs ++ {tid |->
296               mu(tc, joints |-> tc.joints ++ {jid |-> joint})})
297   pre tid in set dom ar.trackcs and
298       let tc = ar.trackcs(tid) in
299           jid not in set dom tc.joints and
300           TD_Used_for_NonInsulated_TrackC(tc.td, tc.atbt,
301               rng tc.joints union {joint}) and
302           Only_One_Next_TrackC(ar.trackcs, tid, jid) and
303           forall tid1 in set dom ar.trackcs & tid1 <> tid =>
304               Joint_and_Next_TrackC(ar.trackcs(tid1),
305               mu(tc,joints |-> tc.joints ++ {jid |-> joint}), jid)
306   post tid in set dom RESULT.trackcs and
307        jid in set dom RESULT.trackcs(tid).joints and
308        dom RESULT.trackcs = dom ar.trackcs and
309        {tid} <-: RESULT.trackcs = {tid} <-: ar.trackcs and
310        RESULT.trackcs(tid) = mu(ar.trackcs(tid),
311            joints |-> ar.trackcs(tid).joints ++ {jid |-> joint}) and
312        RESULT.trackcs(tid).joints(jid) = joint and
313        RESULT.trackcs(tid).atc = ar.trackcs(tid).atc and
314        RESULT.trackcs(tid).td = ar.trackcs(tid).td and
315        RESULT.trackcs(tid).atbt = ar.trackcs(tid).atbt and
316        RESULT.paths = ar.paths and
317        RESULT.routes = ar.routes;
318
319   Del_Joint : Area * TrackC_id * Joint_id -> Area
320   Del_Joint(ar, tcid, jid) ==
321       let tc = ar.trackcs(tcid) in
322           mu(ar, trackcs |-> ar.trackcs ++
323                {tcid |-> mu(tc, joints |-> {jid} <-: tc.joints)})
324   pre tcid in set dom ar.trackcs and
325       jid in set dom ar.trackcs(tcid).joints and
326       card dom ar.trackcs(tcid).joints > 2 and
327       (forall path in set rng ar.paths &
328           path.tc <> tcid or
329           jid <> path.start and jid <> path.endp)
330   post tcid in set dom RESULT.trackcs and
331        dom RESULT.trackcs = dom ar.trackcs and
332        {tcid} <-: RESULT.trackcs = {tcid} <-: ar.trackcs and
333        jid not in set dom RESULT.trackcs(tcid).joints and
334        RESULT.trackcs(tcid) = mu(ar.trackcs(tcid),
335            joints |-> {jid} <-: ar.trackcs(tcid).joints) and
336        RESULT.trackcs(tcid).atc = ar.trackcs(tcid).atc and
337        RESULT.trackcs(tcid).td = ar.trackcs(tcid).td and
338        RESULT.trackcs(tcid).atbt = ar.trackcs(tcid).atbt and
339        RESULT.trackcs(tcid).joints = {jid} <-: ar.trackcs(tcid).joints and
340        RESULT.paths = ar.paths and
341        RESULT.routes = ar.routes;
342
343   Add_Path : Area * Path_id * Path -> Area
344   Add_Path(ar, pid, path) ==
345       mu(ar, paths |-> ar.paths ++ {pid |-> path})
346   pre pid not in set dom ar.paths and
347       Path_within_TrackC(ar.trackcs, path) and
348       Direction_Correct(ar.trackcs, path) and
349       forall p in set rng ar.paths &
350           Not_Same_Path(p, path) and Not_Start_and_End(p, path)
351   post pid in set dom RESULT.paths and
352        RESULT.paths = ar.paths ++ {pid |-> path} and
353        RESULT.paths(pid) = path and
354        RESULT.trackcs = ar.trackcs and
355        RESULT.routes = ar.routes;
356
357   Del_Path : Area * Path_id -> Area
358   Del_Path(ar, pid) ==
359       mu(ar, paths |-> {pid} <-: ar.paths)
360   pre pid in set dom ar.paths and
```

```
361         forall r in set rng ar.routes &        pid not in set elems r.paths
362     post pid not in set dom RESULT.paths and
363          RESULT.paths = {pid} <-: ar.paths and
364          RESULT.trackcs = ar.trackcs and
365          RESULT.routes = ar.routes;
366
367     Add_Route : Area * Route_id * Route -> Area
368     Add_Route (ar, rid, r) ==
369         mu(ar, routes |-> ar.routes ++ {rid |-> r})
370     pre rid not in set dom ar.routes and
371         Path_Exists(ar.paths, r.paths, r.dr) and
372         Exists_ATC_for_Route(ar.trackcs, ar.paths, r) and
373         Route_not_Circular(ar.paths, r) and
374         Path_Connected(ar.paths, r.paths, r.dr)
375     post rid in set dom RESULT.routes and
376          RESULT.routes = ar.routes ++ {rid |-> r} and
377          RESULT.routes(rid) = r and
378          RESULT.trackcs = ar.trackcs and
379          RESULT.paths = ar.paths;
380
381     Del_Route : Area * Route_id -> Area
382     Del_Route(ar, rid) ==
383         mu(ar, routes |-> {rid} <-: ar.routes)
384     pre rid in set dom ar.routes
385     post rid not in set dom RESULT.routes and
386          RESULT.routes = {rid} <-: ar.routes and
387          RESULT.trackcs = ar.trackcs and
388          RESULT.paths = ar.paths;
389
390     Add_Condition : Area * Path_id * Condition -> Area
391     Add_Condition(ar, pid, con) ==
392         let p = mu(ar.paths(pid),
393                    condition |-> ar.paths(pid).condition union {con}) in
394             mu(ar, paths |-> ar.paths ++ {pid |-> p})
395     pre pid in set dom ar.paths and
396         let p = ar.paths(pid) in
397             ar.trackcs(p.tc).joints(p.start).position <= con.start and
398             con.endp <= ar.trackcs(p.tc).joints(p.endp).position and
399             (forall c in set p.condition & Condition_not_Conflict(c, con))
400
401     post pid in set dom RESULT.paths and
402          dom RESULT.paths = dom ar.paths and
403          {pid} <-: RESULT.paths = {pid} <-: ar.paths and
404          RESULT.paths(pid) = mu(ar.paths(pid),
405              condition |-> ar.paths(pid).condition union {con}) and
406          RESULT.paths(pid).start = ar.paths(pid).start and
407          RESULT.paths(pid).endp = ar.paths(pid).endp and
408          RESULT.paths(pid).tc = ar.paths(pid).tc and
409          RESULT.paths(pid).length = ar.paths(pid).length and
410          RESULT.paths(pid).condition = ar.paths(pid).condition union {con} and
411          RESULT.trackcs = ar.trackcs and
412          RESULT.routes = ar.routes;
413
414     Del_Condition : Area * Path_id * Cond_Kind * nat * nat -> Area
415     Del_Condition(ar, pid, kind, start, endp) ==
416             let p = mu(ar.paths(pid), condition |->
417                     {l | l in set ar.paths(pid).condition &
418                      not (l.kind = kind and l.start = start and l.endp = endp)}) in
419             mu(ar, paths |-> ar.paths ++ {pid |-> p})
420     pre     pid in set dom ar.paths and
421             (exists c in set ar.paths(pid).condition &
422                     c.kind = kind and c.start = start and c.endp = endp)
423     post    pid in set dom RESULT.paths and
424             dom RESULT.paths = dom ar.paths and
425             {pid} <-: RESULT.paths = {pid} <-: ar.paths and
426             RESULT.paths(pid) = mu(ar.paths(pid), condition |->
427                    {l | l in set ar.paths(pid).condition &
428                    not (l.kind = kind and l.start = start and l.endp = endp)}) and
429             RESULT.paths(pid).start = ar.paths(pid).start and
430             RESULT.paths(pid).endp = ar.paths(pid).endp and
431             RESULT.paths(pid).tc = ar.paths(pid).tc and
432             RESULT.paths(pid).length = ar.paths(pid).length and
433             RESULT.trackcs = ar.trackcs and
434             RESULT.routes = ar.routes;
435
436     --
437     -- end of Operation (Area level)
```

```
438  --
439
```

B.2.2 基本構造 レベル 2 – 線区に関する定義

構成要素および不変条件

```
440  types
441  --
442  -- Line -- collection of Areas
443  --
444  --
445
446  Line :: areas : Area_map
447          connect : Connect_map
448  --      ver : Version                --
449  --      edit_date : Date             -- These 4 fields are
450  --      valid_date : Date            --       to be used version control system.
451  --      locked : bool                --
452  inv mk_Line(areas, connect) ==
453      forall c in set dom connect &
454          (forall n in set c & Area_Joint_Exists(areas, n)) and
455
456          (forall n1, n2 in set c & n1 <> n2 =>
457              Direction_for_Area_Joint(areas(n1.aid).paths, n1.no,
458                  areas(n2.aid).paths, n2.no, connect(c).chng_direction) and
459              let tc1 = areas(n1.aid).trackcs(n1.tcid),
460                  tc2 = areas(n2.aid).trackcs(n2.tcid) in
461                  Joint_Compatible(tc1.joints(n1.no), tc2.joints(n2.no),
462                                   connect(c)) and
463                  Is_wf_adjacent_signal(tc1, n1.no, tc2, n2.no,
464                                   connect(c).chng_direction));
465
466  Area_map = map Area_id to Area;
467  Area_id = token;
468
469  -- (Plan)
470  -- Area_id :: lid : nat
471  --            aid : nat
472
473  --
474  -- connection between two areas
475  --
476  Area_Joint :: aid : Area_id
477                tcid : TrackC_id
478                no : Joint_id;
479
480  Connect = set of Area_Joint
481  inv con == card con = 2 and
482             forall a1, a2 in set con & a1 <> a2 => a1.aid <> a2.aid;
483
484  Connect_map = map Connect to Remark_Connect
485  inv con == forall a1, a2 in set dom con & a1 <> a2 => a1 inter a2 = {};
486
487  Remark_Connect :: chng_direction : bool
488                    chng_distance : bool;
489
490  functions
491  Area_Joint_Exists : Area_map * Area_Joint -> bool
492  Area_Joint_Exists(areas, n) ==
493      n.aid in set dom areas and
494      n.tcid in set dom areas(n.aid).trackcs and
495      n.no in set dom areas(n.aid).trackcs(n.tcid).joints and
496      not areas(n.aid).trackcs(n.tcid).joints(n.no).remark.line_terminal and
497      forall tcid in set dom areas(n.aid).trackcs & n.tcid <> tcid =>
498          n.no not in set dom areas(n.aid).trackcs(tcid).joints;
499
500  Direction_for_Area_Joint : Path_map * Joint_id * Path_map * Joint_id * bool
501          -> bool
502  Direction_for_Area_Joint(pm1, n1, pm2, n2, chng_dir) ==
503      forall p1 in set rng pm1, p2 in set rng pm2 &
504          ((p1.start = n1 and p2.start = n2) => chng_dir) and
505          ((p1.endp  = n1 and p2.endp  = n2) => chng_dir) and
506          ((p1.start = n1 and p2.endp  = n2) => not chng_dir) and
507          ((p1.endp  = n1 and p2.start = n2) => not chng_dir);
```

```
508
509 Joint_Compatible: Joint * Joint * Remark_Connect -> bool
510 Joint_Compatible(j1, j2, rm) ==
511     j1.insulated = j2.insulated and
512         ((j1.position <> j2.position) = rm.chng_distance) and
513         Remark_Compatible(j1.remark, j2.remark, rm.chng_direction);
514
515 Add_Area : Line * Area_id * Area_Kind * MaxSpeed -> Line
516 Add_Area(ln, aid, kind, max) ==
517     mu(ln, areas |-> ln.areas ++ {aid |-> mk_Area({|->}, {|->}, {|->}, kind, max)})
518 pre aid not in set dom ln.areas
519 post aid in set dom RESULT.areas and
520      RESULT.connect = ln.connect;
521
522 Change_Area : Line * Area_id * Area -> Line
523 Change_Area(ln, aid, area) ==
524     mu(ln, areas |-> ln.areas ++ {aid |-> area})
525 pre aid in set dom ln.areas and
526     inv_Line(mk_Line(ln.areas ++ {aid |-> area}, ln.connect))
527 post RESULT.areas(aid) = area and
528      RESULT.connect = ln.connect;
529
530 Del_Area : Line * Area_id -> Line
531 Del_Area(ln, aid) ==
532     mu(ln, areas |-> {aid} <-: ln.areas)
533 pre aid in set dom ln.areas and
534     (forall c in set dom ln.connect &
535         forall aj in set c & aj.aid <> aid)
536 post aid not in set dom RESULT.areas and
537      RESULT.connect = ln.connect;
538
539 Add_Connect : Line * Connect * Remark_Connect -> Line
540 Add_Connect(ln, con, r) ==
541     mu(ln, connect |-> ln.connect ++ {con |-> r})
542 pre (forall c in set dom ln.connect & c inter con = {}) and
543     (forall n in set con & Area_Joint_Exists(ln.areas, n)) and
544
545     (forall n1, n2 in set con & n1 <> n2 =>
546         Direction_for_Area_Joint(ln.areas(n1.aid).paths, n1.no,
547             ln.areas(n2.aid).paths, n2.no, r.chng_direction) and
548         let tc1 = ln.areas(n1.aid).trackcs(n1.tcid),
549             tc2 = ln.areas(n2.aid).trackcs(n2.tcid) in
550             Joint_Compatible(tc1.joints(n1.no), tc2.joints(n2.no), r) and
551             Is_wf_adjacent_signal(tc1,n1.no, tc2, n2.no, r.chng_direction))
552
553 post con in set dom RESULT.connect and
554      RESULT.connect = ln.connect ++ {con |-> r} and
555      RESULT.connect(con) = r and
556      RESULT.areas = ln.areas;
557
558 Del_Connect : Line * Area_Joint -> Line
559 Del_Connect(ln, n) ==
560     mu(ln, connect |->
561         {c |-> ln.connect(c) | c in set dom ln.connect & n not in set c})
562 pre exists c in set dom ln.connect & n in set c
563 post forall c in set dom RESULT.connect & n not in set c and
564      RESULT.areas = ln.areas;
565
```

線区を操作する関数

```
566 functions
567
568 Line_Add_TrackC : Line * Area_id * TrackC_id * TrackC -> Line
569 Line_Add_TrackC(ln, aid, tcid, tc) ==
570     mu(ln, areas |-> ln.areas ++
571          {aid |-> Add_TrackC(ln.areas(aid), tcid, tc)})
572 pre aid in set dom ln.areas and
573     pre_Add_TrackC(ln.areas(aid), tcid, tc) and
574     (forall jid in set dom tc.joints &
575         forall c in set dom ln.connect &
576             forall n in set c &
577                 n.aid = aid => n.no <> jid)
578 post aid in set dom RESULT.areas and
579      dom ln.areas = dom RESULT.areas and
580      {aid} <-: RESULT.areas = {aid} <-: ln.areas and
```

```
581       tcid in set dom RESULT.areas(aid).trackcs and
582       RESULT.areas(aid).trackcs =
583           ln.areas(aid).trackcs ++ {tcid |-> tc} and
584       RESULT.areas(aid).trackcs(tcid) = tc and
585       RESULT.connect = ln.connect;
586
587   Line_Del_TrackC : Line * Area_id * TrackC_id -> Line
588   Line_Del_TrackC(ln, aid, tcid) ==
589       mu(ln, areas |-> ln.areas ++
590           {aid |-> Del_TrackC(ln.areas(aid), tcid)})
591   pre aid in set dom ln.areas and
592       pre_Del_TrackC(ln.areas(aid), tcid) and
593       forall c in set dom ln.connect &
594           forall aj in set c &
595               aj.aid = aid => aj.tcid <> tcid
596   post aid in set dom RESULT.areas and
597        dom ln.areas = dom RESULT.areas and
598        {aid} <-: RESULT.areas = {aid} <-: ln.areas and
599        RESULT.areas(aid).trackcs = {tcid} <-: ln.areas(aid).trackcs and
600        tcid not in set dom RESULT.areas(aid).trackcs and
601        RESULT.connect = ln.connect;
602
603   Line_Add_Joint : Line * Area_id * TrackC_id * Joint_id * Joint -> Line
604   Line_Add_Joint(ln, aid, tcid, jid, j) ==
605       mu(ln, areas |-> ln.areas ++
606            {aid |-> Add_Joint(ln.areas(aid), tcid, jid, j)})
607   pre aid in set dom ln.areas and
608       pre_Add_Joint(ln.areas(aid), tcid, jid, j) and
609       (forall c in set dom ln.connect &
610           forall n in set c &
611               n.aid = aid => n.no <> jid)
612   post aid in set dom RESULT.areas and
613        tcid in set dom RESULT.areas(aid).trackcs and
614        jid in set dom RESULT.areas(aid).trackcs(tcid).joints and
615        dom RESULT.areas = dom ln.areas and
616        {aid} <-: RESULT.areas = {aid} <-: ln.areas and
617        dom RESULT.areas(aid).trackcs = dom ln.areas(aid).trackcs and
618        {tcid} <-: RESULT.areas(aid).trackcs =
619            {tcid} <-: ln.areas(aid).trackcs and
620        RESULT.areas(aid).trackcs(tcid).joints =
621            ln.areas(aid).trackcs(tcid).joints ++ {jid |-> j} and
622        RESULT.areas(aid).trackcs(tcid).joints(jid) = j and
623        RESULT.connect = ln.connect;
624
625   Line_Del_Joint : Line * Area_id * TrackC_id * Joint_id -> Line
626   Line_Del_Joint(ln, aid, tcid, jid) ==
627       mu(ln, areas |-> ln.areas ++
628           {aid |-> Del_Joint(ln.areas(aid), tcid, jid)})
629   pre aid in set dom ln.areas and
630       pre_Del_Joint(ln.areas(aid), tcid, jid) and
631       forall c in set dom ln.connect &
632           forall aj in set c &
633               (aj.aid = aid and aj.tcid = tcid) => aj.no <> jid
634   post aid in set dom RESULT.areas and
635        tcid in set dom RESULT.areas(aid).trackcs and
636        jid not in set dom RESULT.areas(aid).trackcs(tcid).joints and
637        dom RESULT.areas = dom ln.areas and
638        {aid} <-: RESULT.areas = {aid} <-: ln.areas and
639        dom RESULT.areas(aid).trackcs = dom ln.areas(aid).trackcs and
640        {tcid} <-: RESULT.areas(aid).trackcs =
641            {tcid} <-: ln.areas(aid).trackcs and
642        RESULT.areas(aid).trackcs(tcid).joints =
643            {jid} <-: ln.areas(aid).trackcs(tcid).joints and
644        RESULT.connect = ln.connect;
645
646   Line_Add_Path : Line * Area_id * Path_id * Path -> Line
647   Line_Add_Path(ln, aid, pid, p) ==
648       mu(ln, areas |-> ln.areas ++
649           {aid |-> Add_Path(ln.areas(aid), pid, p)})
650   pre aid in set dom ln.areas and
651       pre_Add_Path(ln.areas(aid), pid, p) and
652       forall c in set dom ln.connect &
653           forall n1, n2 in set c &
654               (n1 <> n2 and n1.aid = aid) =>
655                    Direction_for_Area_Joint({pid |-> p}, n1.no,
656                     ln.areas(n2.aid).paths , n2.no, ln.connect(c).chng_direction)
657   post aid in set dom RESULT.areas and
```

```
658         pid in set dom RESULT.areas(aid).paths and
659         dom RESULT.areas = dom ln.areas and
660         {aid} <-: RESULT.areas = {aid} <-: ln.areas and
661         RESULT.areas(aid).paths = ln.areas(aid).paths ++ {pid |-> p} and
662         RESULT.areas(aid).paths(pid) = p and
663         RESULT.connect = ln.connect;
664
665   Line_Del_Path : Line * Area_id * Path_id -> Line
666   Line_Del_Path(ln, aid, pid) ==
667       mu(ln, areas |-> ln.areas ++ {aid |-> Del_Path(ln.areas(aid), pid)})
668   pre aid in set dom ln.areas and
669       pre_Del_Path(ln.areas(aid), pid)
670   post aid in set dom RESULT.areas and
671         pid not in set dom RESULT.areas(aid).paths and
672         dom RESULT.areas = dom ln.areas and
673         {aid} <-: RESULT.areas = {aid} <-: ln.areas and
674         RESULT.areas(aid).paths = {pid} <-: ln.areas(aid).paths and
675         RESULT.connect = ln.connect;
676
677   Line_Add_Route : Line * Area_id * Route_id * Route -> Line
678   Line_Add_Route(ln, aid, rid, r) ==
679       mu(ln, areas |-> ln.areas ++ {aid |-> Add_Route(ln.areas(aid), rid,r)})
680   pre aid in set dom ln.areas and
681       pre_Add_Route(ln.areas(aid), rid, r)
682   post aid in set dom RESULT.areas and
683         rid in set dom RESULT.areas(aid).routes and
684         dom RESULT.areas = dom ln.areas and
685         {aid} <-: RESULT.areas = {aid} <-: ln.areas and
686         RESULT.areas(aid).routes = ln.areas(aid).routes ++ {rid |-> r} and
687         RESULT.areas(aid).routes(rid) = r and
688         RESULT.connect = ln.connect;
689
690   Line_Del_Route : Line * Area_id * Route_id -> Line
691   Line_Del_Route(ln, aid, rid) ==
692       mu(ln, areas |-> ln.areas ++ {aid |-> Del_Route(ln.areas(aid), rid)})
693   pre aid in set dom ln.areas and
694       pre_Del_Route(ln.areas(aid), rid)
695   post aid in set dom RESULT.areas and
696         dom RESULT.areas = dom ln.areas and
697         {aid} <-: RESULT.areas = {aid} <-: ln.areas and
698         rid not in set dom RESULT.areas(aid).routes and
699         RESULT.areas(aid).routes = {rid} <-: ln.areas(aid).routes and
700         RESULT.connect = ln.connect;
701
702   Line_Add_Condition : Line * Area_id * Path_id * Condition-> Line
703   Line_Add_Condition(ln, aid, pid, con) ==
704       mu(ln, areas |-> ln.areas ++
705                           {aid |-> Add_Condition(ln.areas(aid), pid, con)})
706   pre aid in set dom ln.areas and
707       pre_Add_Condition(ln.areas(aid), pid, con);
708
709   Line_Del_Condition : Line * Area_id * Path_id * Cond_Kind * nat * nat-> Line
710   Line_Del_Condition(ln, aid, pid, kind, start, endp) ==
711       mu(ln, areas |-> ln.areas ++
712            {aid |-> Del_Condition(ln.areas(aid), pid, kind, start, endp)})
713   pre aid in set dom ln.areas and
714       pre_Del_Condition(ln.areas(aid), pid, kind, start, endp);
715
```

B.2.3　完成した（運用に使用できる）データベース構造の条件

```
716   --
717   --  Digital ATC Database
718   --  Condition for "Completed" Database
719   --
720
721   functions
722   Is_wf_Line_DB : Line -> bool
723   Is_wf_Line_DB(ln) ==
724
725       (forall aid in set dom ln.areas & let ar = ln.areas(aid) in
726           Joint_Completed(ar.trackcs, aid, ln.connect) and
727           Path_Exists_for_Joint(ar.trackcs, ar.paths) and
728           Path_Exists_for_TrackC(ar.trackcs, ar.paths) and
729           Route_Exists_for_Path(ar) and
```

```
730         Path_Exists_before_Start(ar, aid, ln.connect) and
731         Path_Exists_after_End(ar, aid, ln.connect) and
732         Route_Exists_to_Terminal(ar, aid, ln.connect) and
733             (ar.kind = <PLAIN> => Is_Plain_Area(ar, aid, ln.connect))) and
734
735         Following_Path_Exists_at_Connect(ln) and
736         Preceding_Path_Exists_at_Connect(ln) and
737         One_Side_Unique_Path_at_Connection(ln);
738
739 Joint_Completed : TrackC_map * Area_id * Connect_map -> bool
740 Joint_Completed(trackcs, aid, connect) ==
741     forall tid in set dom trackcs &
742         let tc = trackcs(tid) in
743             forall jid in set dom tc.joints &
744                 (not exists tcid in set dom trackcs &
745                      tcid <> tid and
746                      jid in set dom trackcs(tcid).joints) =>
747                 (mk_Area_Joint(aid, tid, jid) in set dunion dom connect or
748                  tc.joints(jid).remark.line_terminal);
749
750 Path_Exists_for_Joint : TrackC_map * Path_map -> bool
751 Path_Exists_for_Joint(trackcs, paths) ==
752     forall tid in set dom trackcs &
753         forall jid in set dom trackcs(tid).joints &
754             (exists p in set rng paths &
755                  p.tc = tid and
756                  (p.start = jid or p.endp = jid));
757
758 Path_Exists_for_TrackC : TrackC_map * Path_map -> bool
759 Path_Exists_for_TrackC(trackcs, paths) ==
760     forall tid in set dom trackcs &
761         forall dr in set dom trackcs(tid).atc &
762             trackcs(tid).atc(dr).used =>
763                  exists p in set rng paths &
764                      p.tc = tid and p.used(dr);
765
766 Route_Exists_for_Path : Area -> bool
767 Route_Exists_for_Path(ar) ==
768     forall pid in set dom ar.paths &
769         forall dr in set dom ar.paths(pid).used &
770             ar.paths(pid).used(dr) =>
771                 ar.trackcs(ar.paths(pid).tc).atc(dr).used =>
772                     exists r in set rng ar.routes &
773                         r.dr = dr and pid in set elems r.paths;
774
775 Path_Exists_before_Start :  Area * Area_id * Connect_map -> bool
776 Path_Exists_before_Start(ar, aid, connect) ==
777     forall pid in set dom ar.paths &
778         let p = ar.paths(pid) in
779             forall dr in set dom p.used &
780                 p.used(dr) =>
781                 (mk_Area_Joint(aid, p.tc, p.start) in set dunion dom connect or
782                 ar.trackcs(p.tc).joints(p.start).remark.line_terminal or
783                 ar.trackcs(p.tc).joints(p.start).remark.atc_terminal(dr) or
784                 exists pid1 in set dom ar.paths &
785                     let p1 = ar.paths(pid1) in
786                         p1.tc <> p.tc and
787                         p1.used(dr) and
788                         p1.endp = p.start);
789
790 Path_Exists_after_End :  Area * Area_id * Connect_map -> bool
791 Path_Exists_after_End(ar, aid, connect) ==
792     forall pid in set dom ar.paths &
793        let p = ar.paths(pid) in
794        forall dr in set dom p.used &
795            p.used(dr) =>
796                 (mk_Area_Joint(aid, p.tc, p.endp) in set dunion dom connect or
797                 ar.trackcs(p.tc).joints(p.endp).remark.line_terminal or
798                 ar.trackcs(p.tc).joints(p.endp).remark.atc_terminal(dr) or
799                 exists pid1 in set dom ar.paths &
800                     let p1 = ar.paths(pid1) in
801                         p1.tc <> p.tc and
802                         p1.used(dr) and
803                         p1.start = p.endp);
804
805 StartJoint : Path * Direction -> Joint_id
806 StartJoint(path, dr) ==
```

```
807             if dr = <ADIR> then path.start else path.endp
808     post (dr = <ADIR> => RESULT = path.start) and
809          (dr = <BDIR> => RESULT = path.endp);
810
811     EndJoint : Path * Direction -> Joint_id
812     EndJoint(path, dr) ==
813             if dr = <ADIR> then path.endp else path.start
814     post (dr = <ADIR> => RESULT = path.endp) and
815          (dr = <BDIR> => RESULT = path.start);
816
817     --
818     -- Route_Exists_to_Terminal means that Train can reach an Area_Joint or
819     --                                                  an end of track.
820     Route_Exists_to_Terminal : Area * Area_id * Connect_map -> bool
821     Route_Exists_to_Terminal(ar, aid, connect) ==
822         forall rid in set dom ar.routes &
823             let r = ar.routes(rid) in
824             let pid = r.paths(len r.paths) in
825             let jid = EndJoint(ar.paths(pid), r.dr),
826                 tcid = ar.paths(pid).tc in
827
828                 mk_Area_Joint(aid, tcid, jid) in set dunion dom connect or
829                 ar.trackcs(tcid).joints(jid).remark.line_terminal or
830                 ar.trackcs(tcid).joints(jid).remark.atc_terminal(r.dr) or
831                 Following_Route_Exists(ar.routes, rid) or
832                 Following_Path_Unique(ar.paths, pid, r.dr);
833
834     --
835     -- Following_Route_Exists1 :
836     -- On the last path, if a next route which includes following paths can
837     -- be indicated, train can proceed with next route ID.
838     --
839
840     Following_Route_Exists : Route_map * Route_id -> bool
841     Following_Route_Exists(routes, rid) ==
842         exists rid1 in set dom routes &
843             let r = routes(rid), r1 = routes(rid1) in
844                 r1.dr = r.dr and
845                 exists i in set inds r1.paths &
846                     r1.paths(i) = r.paths(len r.paths) and
847                     i < len r1.paths
848     pre rid in set dom routes;
849
850     --
851     -- Unique_Next_Path :
852     -- On the last path, if there is only one next path possible,
853     --  trains can proceed to the next path.
854     --
855     Following_Path_Unique : Path_map * Path_id * Direction-> bool
856     Following_Path_Unique(paths, pid, dr) ==
857         exists1 pid1 in set dom paths &
858             paths(pid1).tc <> paths(pid).tc and
859             paths(pid1).used(dr) and
860             EndJoint(paths(pid), dr) = StartJoint(paths(pid1), dr)
861     pre pid in set dom paths;
862
863     --
864     -- Plain Area, where from TrackC_id and direction, Route can be determined.
865     --
866
867     Is_Plain_Area : Area * Area_id * Connect_map-> bool
868     Is_Plain_Area(ar, aid, connect) ==
869         (forall tcid in set dom ar.trackcs &
870             forall dr in set dom ar.trackcs(tcid).atc &
871                 ar.trackcs(tcid).atc(dr).used =>
872                     exists1 rid in set dom ar.routes &
873                         ar.routes(rid).dr = dr and
874                         exists pid in set elems ar.routes(rid).paths &
875                             ar.paths(pid).tc = tcid) and
876             (forall r in set rng ar.routes &
877                 let p = ar.paths(r.paths(len r.paths)) in
878                 let jid = EndJoint(p, r.dr) in
879                     mk_Area_Joint(aid, p.tc, jid) in set dunion dom connect or
880                     ar.trackcs(p.tc).joints(jid).remark.line_terminal or
881                     ar.trackcs(p.tc).joints(jid).remark.atc_terminal(r.dr));
882
883     --
```

```
884  -- One_Side_Unique_Path_at_Connection
885  --  At the connection it is not favorable that
886  --  in both area paths are not unique.
887  -- Because it makes imposibble to indicate next path
888  --                              in the former track circuit.
889
890  One_Side_Unique_Path_at_Connection : Line -> bool
891  One_Side_Unique_Path_at_Connection(ln) ==
892      forall con in set dom ln.connect &
893          forall n1, n2 in set con & n1 <> n2 =>
894              forall dr in set {<ADIR>, <BDIR>} &
895                  card {p | p in set rng ln.areas(n1.aid).paths &
896                      p.used(dr) and
897                      EndJoint(p, dr) = n1.no} > 1 =>
898
899                  (ln.areas(n1.aid).trackcs(n1.tcid).joints(n1.no).remark.atc_terminal(dr) or
900
901                   let dr2 = if not ln.connect(con).chng_direction then dr
902                             else if dr = <ADIR> then <BDIR> else <ADIR> in
903                       card {p | p in set rng ln.areas(n2.aid).paths &
904                                 p.used(dr2) and
905                                 StartJoint(p, dr2) = n2.no} = 1);
906
907  Following_Path_Exists_at_Connect : Line -> bool
908  Following_Path_Exists_at_Connect(ln) ==
909      forall con in set dom ln.connect &
910          forall n1, n2 in set con & n1 <> n2 =>
911              forall dr in set {<ADIR>, <BDIR>} &
912                  (exists p in set rng ln.areas(n1.aid).paths &
913                       p.used(dr) and
914                       EndJoint(p, dr) = n1.no) =>
915                  (ln.areas(n1.aid).trackcs(n1.tcid).joints(n1.no).remark.atc_terminal(dr) or
916                  (exists p2 in set rng ln.areas(n2.aid).paths &
917                       let dr2 = if not ln.connect(con).chng_direction then dr
918                                 else if dr = <ADIR> then <BDIR> else <ADIR> in
919                           p2.used(dr2) and
920                           StartJoint(p2, dr2) = n2.no));
921
922  Preceding_Path_Exists_at_Connect : Line -> bool
923  Preceding_Path_Exists_at_Connect(ln) ==
924      forall con in set dom ln.connect &
925          forall n1, n2 in set con & n1 <> n2 =>
926              forall dr in set {<ADIR>, <BDIR>} &
927                  (exists p in set rng ln.areas(n1.aid).paths &
928                       p.used(dr) and
929                       StartJoint(p, dr) = n1.no) =>
930                  (ln.areas(n1.aid).trackcs(n1.tcid).joints(n1.no).remark.atc_terminal(dr) or
931                  (exists p2 in set rng ln.areas(n2.aid).paths &
932                       let dr2 = if not ln.connect(con).chng_direction then dr
933                                 else if dr = <ADIR> then <BDIR> else <ADIR> in
934                           p2.used(dr2) and
935                           EndJoint(p2, dr2) = n2.no));
```

付録 C

単線自動閉そく装置モデル（B：抽象機械のみ）

C.1 自動閉そく（特殊）

C.1.1 AUTOBLOCKNN マシン（全体定義）

```
1   MACHINE
2       AUTO_BLOCKNN
3
4   INCLUDES
5       OE.OriginEquip,
6       DE.DestEquip,
7       DC.DirectCircuit,
8       Const
9
10  INVARIANT
11      OE.direction = DC.directions(origin) &
12      DE.direction = DC.directions(dest) &
13      OE.oppositedirect = DC.directions(dest) &
14      DE.oppositedirect = DC.directions(origin) &
15      OE.existence = DC.exist &
16      DE.existence = DC.exist
17
18  INITIALISATION
19      ANY o_r_le, o_r_lo, o_d_le, o_dir, o_d_lo,
20          d_r_le, d_r_lo, d_d_le, d_dir, d_d_lo, ex WHERE
21
22          o_r_le : LEVER & o_r_lo : LOCK & o_d_le :DIRECTION &
23          o_dir : DIRECTION & o_d_lo : LOCK &
24          d_r_le : LEVER & d_r_lo : LOCK & d_d_le :DIRECTION &
25          d_dir : DIRECTION & d_d_lo : LOCK & ex : EXIST &
26
27          not(o_dir = down & d_dir = up) &
28          (o_dir = up => (o_r_lo = unlock & o_d_lo = unlock)) &
29          (o_dir = down =>
30              (((ex = detect or o_r_lo = lock or o_r_le = set) => o_d_lo = lock) &
31               ((ex = nondetect & o_r_lo = unlock & o_r_le = unset) => o_d_lo = unlock))) &
32
33          (d_dir = down => (d_r_lo = unlock & d_d_lo = unlock)) &
34          (d_dir = up =>
35              (((ex = detect or d_r_lo = lock or d_r_le = set) => d_d_lo = lock) &
36               ((ex = nondetect & d_r_lo = unlock & d_r_le = unset) => d_d_lo = unlock)))
37      THEN
38              OE.PSet(o_r_le, o_r_lo, o_d_le, o_dir, o_d_lo, d_dir, ex) ||
39              DE.PSet(d_r_le, d_r_lo, d_d_le, d_dir, d_d_lo, o_dir, ex) ||
40              DC.Set(o_dir, d_dir, ex)
41      END
42
43  OPERATIONS
44      xx <-- OriginDownSet =
45      BEGIN
46          xx <-- OE.DownSet ||
```

```
47          IF OE.direction = up & OE.existence = nondetect & OE.oppositedirect = down THEN
48              DE.OppositeDirectChange(down)||
49              DC.DirectionSet(origin, down)
50          END
51      END;
52
53      xx <-- OriginUpSet =
54      BEGIN
55          xx <-- OE.UpSet ||
56          IF OE.direction = down & OE.directlock = unlock THEN
57              DE.OppositeDirectChange(up) ||
58              DC.DirectionSet(origin, up)
59          END
60      END;
61
62      xx <-- DestUpSet =
63      BEGIN
64          xx <-- DE.UpSet ||
65          IF DE.direction = down & DE.existence = nondetect & DE.oppositedirect = up THEN
66              OE.OppositeDirectChange(up)||
67              DC.DirectionSet(dest, up)
68          END
69      END;
70
71      xx <-- DestDownSet =
72      BEGIN
73          xx <-- DE.DownSet ||
74          IF DE.direction = up & DE.directlock = unlock THEN
75              OE.OppositeDirectChange(down) ||
76              DC.DirectionSet(dest, down)
77          END
78      END;
79
80
81      TrackCDown =
82      PRE DC.exist = nondetect
83      THEN
84          DC.TrackCDown ||
85          OE.TrackCDown ||
86          DE.TrackCDown
87      END;
88
89      xx, yy <-- TrackCUp =
90      PRE DC.exist = detect
91      THEN
92          SELECT OE.direction = down & OE.routelever = unset & OE.routelock = unlock & OE.directlever = up
93          THEN
94              DC.Set(up, down, nondetect) ||
95              xx <-- OE.TrackCUp ||
96              DE.PSet(DE.routelever, DE.routelock, DE.directlever,
97                      DE.direction, DE.directlock, up, nondetect) ||
98              yy := down
99          WHEN DE.direction = up & DE.routelever = unset & DE.routelock = unlock & DE.directlever = down
100         THEN
101             xx := up ||
102             DC.Set(up, down, nondetect) ||
103             yy <-- DE.TrackCUp ||
104             OE.PSet(OE.routelever, OE.routelock, OE.directlever,
105                     OE.direction, OE.directlock, down, nondetect)
106         ELSE
107             DC.TrackCUp ||
108             xx <-- OE.TrackCUp ||
109             yy <-- DE.TrackCUp
110         END
111     END;
112
113     OriginRouteSet =
114     PRE OE.routelever = unset
115     THEN
116         OE.RouteSet
117     END;
118
119     xx <-- OriginRouteRelease =
120     PRE OE.routelever = set
121     THEN
122         xx <-- OE.RouteRelease ||
123         IF OE.direction = down & OE.existence = nondetect & OE.routelock = unlock & OE.directlever = up
```

```
124             THEN
125                 DC.DirectionSet(origin, up) ||
126                 DE.OppositeDirectChange(up)
127             END
128         END;
129
130         DestRouteSet =
131         PRE DE.routelever = unset
132         THEN
133             DE.RouteSet
134         END;
135
136         xx <-- DestRouteRelease =
137         PRE DE.routelever = set
138         THEN
139             xx <-- DE.RouteRelease ||
140             IF DE.direction = up & DE.existence = nondetect & DE.routelock = unlock & DE.directlever = down
141             THEN
142                 DC.DirectionSet(dest, down) ||
143                 OE.OppositeDirectChange(down)
144             END
145         END;
146
147         OriginRouteLock =
148         PRE OE.routelock = unlock & OE.routelever = set & OE.direction = down
149         THEN
150             OE.RouteLock
151         END;
152
153         DestRouteLock =
154         PRE DE.routelock = unlock & DE.routelever = set & DE.direction = up
155         THEN
156             DE.RouteLock
157         END;
158
159         xx <-- OriginRouteUnLock =
160         PRE OE.routelock = lock & OE.direction = down & xx : DIRECTION
161         THEN
162            xx <-- OE.RouteUnLock ||
163            IF OE.routelever = unset & OE.existence = nondetect & OE.directlever = up THEN
164               DC.DirectionSet(origin, up) ||
165               DE.OppositeDirectChange(up)
166            END
167         END;
168
169         xx <-- DestRouteUnLock =
170         PRE DE.routelock = lock & DE.direction = up & xx : DIRECTION
171         THEN
172            xx <-- DE.RouteUnLock ||
173            IF DE.routelever = unset & DE.existence = nondetect & DE.directlever = down THEN
174               DC.DirectionSet(dest, down) ||
175               OE.OppositeDirectChange(down)
176            END
177         END
178
179     END
```

C.1.2 Const マシン（定数）

```
1   MACHINE
2       Const
3
4   SETS
5       ASPECT = {green, yellow, red};
6       EXIST = {detect, nondetect};
7       LOCK = {lock, unlock};
8       LEVER = {set, unset};
9       DIRECTION = {up, down};
10      LOCATION = {origin, dest}
11
12  END
```

C.1.3　DirectCircuit マシン（方向回線）

```
MACHINE
    DirectCircuit

SEES
    Const

VARIABLES
    directions, exist

INVARIANT
    directions : LOCATION --> DIRECTION &
    not(directions(origin) = down & directions(dest) = up) &
    exist : EXIST

INITIALISATION
    directions :(directions : LOCATION --> DIRECTION &
                 not(directions(origin) = down & directions(dest) = up)) ||
    exist :: EXIST

OPERATIONS
    TrackCDown =
    PRE exist = nondetect
    THEN
        exist := detect
    END;

    TrackCUp =
    PRE exist = detect
    THEN
        exist := nondetect
    END;

    DirectionSet(loc, dir) =
    PRE loc : LOCATION & dir : DIRECTION
    THEN
        IF
            exist = nondetect &
            not(loc = origin & dir = down & directions(dest) = up) &
            not(loc = dest & dir = up & directions(origin) = down)
        THEN
            directions := directions <+ {loc |-> dir}
        ELSE
            skip
        END
    END;

    Set(loc1, loc2, ex) =
    PRE loc1 : DIRECTION & loc2 : DIRECTION & ex : EXIST &
        not(loc1 = down & loc2 = up)
    THEN
        directions := {origin|-> loc1, dest |-> loc2} ||
        exist := ex
    END

END
```

C.1.4　StationEquip マシン（駅装置）

```
MACHINE
    StationEquip

SEES
    Const

CONCRETE_VARIABLES
    routelever, routelock, directlever, direction, directlock, oppositedirect, existence

INVARIANT
    routelever : LEVER &
    routelock : LOCK &
```

```
13      directlever : DIRECTION &
14      direction : DIRECTION &
15      directlock : LOCK &
16      oppositedirect : DIRECTION &
17      existence : EXIST
18
19  INITIALISATION
20      routelever :: LEVER ||
21      routelock :: LOCK ||
22      directlever :: DIRECTION ||
23      direction :: DIRECTION ||
24      directlock :: LOCK ||
25      oppositedirect :: DIRECTION ||
26      existence :: EXIST
27
28  OPERATIONS
29      Init =
30      BEGIN
31          routelever :: LEVER ||
32          routelock :: LOCK ||
33          directlever :: DIRECTION ||
34          direction :: DIRECTION ||
35          directlock :: LOCK ||
36          oppositedirect :: DIRECTION ||
37          existence :: EXIST
38      END;
39
40      Set(r_le, r_lo, d_le, dir, d_lo, o_dir ,ex) =
41      PRE r_le : LEVER & r_lo : LOCK & d_le : DIRECTION & dir : DIRECTION & d_lo : LOCK &
42          o_dir : DIRECTION & ex : EXIST
43      THEN
44          routelever := r_le ||
45          routelock := r_lo ||
46          directlever := d_le ||
47          direction := dir ||
48          directlock := d_lo ||
49          oppositedirect := o_dir ||
50          existence := ex
51      END
52
53  END
```

C.1.5 OriginEquip マシン（起点方駅装置）

```
1   MACHINE
2       OriginEquip
3
4   INCLUDES
5       StationEquip
6
7   SEES
8       Const
9
10  DEFINITIONS
11      transition(dir, d_lo, op, dir2, d_lo2, op2) ==
12          ((dir = down & op = down) =>
13                (((d_lo = lock & d_lo2 = lock) => dir2 = dir) &
14                 op2 = op)) &
15          ((dir = up & op = up) => dir2 = dir)
16
17  INVARIANT
18      not(direction = down & oppositedirect = up) &
19      (direction = up => (routelock = unlock & directlock = unlock)) &
20      (direction = down =>
21          (((existence = detect or routelock = lock or routelever = set) => directlock = lock) &
22           ((existence = nondetect & routelock = unlock & routelever = unset)=> directlock = unlock)))
23
24  INITIALISATION
25      ANY r_le, r_lo, d_le, dir, d_lo, op, ex WHERE
26          r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
27          dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
28          not(dir = down & op = up) &
29          (dir = up => (r_lo = unlock & d_lo = unlock)) &
30          (dir = down =>
31              (((ex = detect or r_lo = lock or r_le = set) => d_lo = lock)) &
```

```
32            ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))
33        THEN
34            Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
35        END
36
37    OPERATIONS
38        xx <-- DownSet =
39        ANY dir, dir_l WHERE
40            dir : DIRECTION & dir_l :LOCK &
41            ((direction = up & existence = nondetect & oppositedirect = down) =>
42                (dir = down &
43                 (routelever = set => dir_l = lock) &
44                 (routelever = unset => dir_l = unlock))) &
45            ((direction = down or existence = detect or oppositedirect = up) =>
46                (dir = direction & dir_l = directlock)) &
47            transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
48        THEN
49            Set(routelever, routelock, down, dir, dir_l, oppositedirect ,existence) ||
50            xx := dir
51        END;
52
53
54        xx <-- UpSet =
55        ANY dir WHERE
56            dir : DIRECTION &
57            ((direction = down & directlock = unlock) => dir = up) &
58            ((direction = up or directlock = lock) => dir = direction) &
59            transition(direction, directlock, oppositedirect, dir, directlock, oppositedirect)
60        THEN
61            Set(routelever, routelock, up, dir, directlock, oppositedirect ,existence) ||
62            xx := dir
63        END;
64
65        OppositeDirectChange(dir) =
66        PRE dir : DIRECTION
67        THEN
68            IF direction = up THEN
69                Set(routelever, routelock, directlever, direction, directlock, dir, existence)
70            ELSE
71                skip
72            END
73        END;
74
75        TrackCDown =
76        PRE existence = nondetect
77        THEN
78            ANY dir_l WHERE
79                dir_l : LOCK &
80                (direction = down => dir_l = lock) &
81                (direction = up => dir_l = directlock) &
82                transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
83            THEN
84                Set(routelever, routelock, directlever, direction, dir_l, oppositedirect, detect)
85            END
86        END;
87
88        xx <-- TrackCUp =
89        PRE existence = detect
90        THEN
91            ANY dir_l, dir WHERE
92                dir_l : LOCK & dir : DIRECTION &
93                ((direction = down & routelever = unset & routelock = unlock)
94                    => (dir_l = unlock & dir = directlever)) &
95                ((direction = up or routelever = set or routelock = lock)
96                    => (dir_l = directlock & dir = direction)) &
97                transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
98            THEN
99                xx := dir ||
100               Set(routelever, routelock, directlever, dir, dir_l, oppositedirect, nondetect)
101           END
102       END;
103
104       RouteSet =
105       PRE routelever = unset
106       THEN
107           ANY dir_l WHERE
108               dir_l : LOCK &
```

```
109             (direction = down => dir_l = lock) &
110             (direction = up => dir_l = directlock) &
111             transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
112
113         THEN
114            Set(set, routelock, directlever, direction, dir_l, oppositedirect, existence)
115         END
116     END;
117
118     xx <-- RouteRelease =
119     PRE routelever = set
120     THEN
121         ANY dir, dir_l WHERE
122             dir : DIRECTION & dir_l : LOCK &
123             ((direction = down & existence = nondetect & routelock = unlock) =>
124                 (dir = directlever & dir_l = unlock)) &
125             ((direction = up or existence = detect or routelock = lock) =>
126                 (dir = direction & dir_l = directlock)) &
127             transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
128         THEN
129             xx := dir ||
130             Set(unset, routelock, directlever, dir, dir_l, oppositedirect, existence)
131         END
132     END;
133
134     RouteLock =
135     PRE routelock = unlock & routelever = set & direction = down
136     THEN
137         Set(routelever, lock, directlever, direction, directlock, oppositedirect, existence)
138     END;
139
140     xx <-- RouteUnLock =
141     PRE routelock = lock & direction = down
142     THEN
143         ANY dir, dir_l WHERE
144             dir : DIRECTION & dir_l : LOCK &
145             ((routelever = unset & existence = nondetect) => (dir = directlever & dir_l = unlock)) &
146             ((routelever = set or existence = detect) => (dir = direction & dir_l = directlock)) &
147             transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
148         THEN
149             xx := dir ||
150             Set(routelever, unlock, directlever, dir, dir_l, oppositedirect, existence)
151         END
152     END;
153
154     PSet(r_le, r_lo, d_le, dir, d_lo, op, ex) =
155     PRE
156         r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
157         dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
158         not(dir = down & op = up) &
159         (dir = up => (r_lo = unlock & d_lo = unlock)) &
160         (dir = down =>
161             (((ex = detect or r_lo = lock or r_le = set) => d_lo = lock) &
162              ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock)))
163     THEN
164         Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
165     END
166
167 END
```

C.1.6 DestEquip マシン（終点方駅装置）

```
1  MACHINE
2      DestEquip
3
4  INCLUDES
5      StationEquip
6
7  SEES
8      Const
9
10 DEFINITIONS
11     transition(dir, d_lo, op, dir2, d_lo2, op2) ==
12         ((dir = up & op = up) =>
13               (((d_lo = lock & d_lo2 = lock) => dir2 = dir) &
```

```
                  op2 = op)) &
        ((dir = down & op = down) => dir2 = dir)

INVARIANT
    not(direction = up & oppositedirect = down) &
    (direction = down => (routelock = unlock & directlock = unlock)) &
    (direction = up =>
        (((existence = detect or routelock = lock or routelever = set) => directlock = lock) &
        ((existence = nondetect & routelock = unlock & routelever = unset) => directlock = unlock)))

INITIALISATION
    ANY r_le, r_lo, d_le, dir, d_lo, op, ex WHERE
        r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
        dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
        not(dir = up & op = down) &
        (dir = down => (r_lo = unlock & d_lo = unlock)) &
        (dir = up =>
             (((ex = detect or r_lo = lock or r_le = set) => d_lo = lock)) &
              ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))
    THEN
        Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
    END

OPERATIONS
    xx <-- UpSet =
    ANY dir, d_l WHERE
        dir : DIRECTION & d_l :LOCK &
        ((direction = down & existence = nondetect & oppositedirect = up) =>
            (dir = up &
             (routelever = set => d_l = lock) &
             (routelever = unset => d_l = unlock))) &
        ((direction = up or existence = detect or oppositedirect = down) =>
            (dir = direction & d_l = directlock)) &
        transition(direction, directlock, oppositedirect, dir, d_l, oppositedirect)
    THEN
        Set(routelever, routelock, up, dir, d_l, oppositedirect ,existence) ||
        xx := dir
    END;


    xx <-- DownSet =
    ANY dir WHERE
        dir  : DIRECTION &
        ((direction = up & directlock = unlock) => dir = down) &
        ((direction = down or directlock = lock) => dir = direction) &
        transition(direction, directlock, oppositedirect, dir, directlock, oppositedirect)
    THEN
        Set(routelever, routelock, down, dir, directlock, oppositedirect ,existence) ||
        xx := dir
    END;

    OppositeDirectChange(dir) =
    PRE dir : DIRECTION
    THEN
        IF direction = down THEN
            Set(routelever, routelock, directlever, direction, directlock, dir ,existence)
        ELSE
            skip
        END
    END;

    TrackCDown =
    PRE existence = nondetect
    THEN
        ANY dir_l WHERE
            dir_l : LOCK &
            (direction = up => dir_l = lock) &
            (direction = down => dir_l = directlock) &
            transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
        THEN
            Set(routelever, routelock, directlever, direction, dir_l, oppositedirect, detect)
        END
    END;

    xx <-- TrackCUp =
    PRE existence = detect
    THEN
```

```
91          ANY dir_l, dir WHERE
92              dir_l : LOCK & dir : DIRECTION &
93              ((direction = up & routelever = unset & routelock = unlock)
94                  => (dir_l = unlock & dir = directlever)) &
95              ((direction = down or routelever = set or routelock = lock)
96                  => (dir_l = directlock & dir = direction)) &
97              transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
98          THEN
99              xx := dir ||
100             Set(routelever, routelock, directlever, dir, dir_l, oppositedirect, nondetect)
101         END
102     END;
103
104     RouteSet =
105     PRE routelever = unset
106     THEN
107         ANY dir_l WHERE
108             dir_l : LOCK &
109             (direction = up => dir_l = lock) &
110             (direction = down => dir_l = directlock) &
111             transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
112         THEN
113            Set(set, routelock, directlever, direction, dir_l, oppositedirect, existence)
114         END
115     END;
116
117     xx <-- RouteRelease =
118     PRE routelever = set
119     THEN
120         ANY dir, dir_l  WHERE
121             dir : DIRECTION & dir_l : LOCK &
122             ((direction = up & existence = nondetect & routelock = unlock) =>
123                 (dir = directlever & dir_l = unlock)) &
124             ((direction = down or existence = detect or routelock = lock) =>
125                 (dir = direction & dir_l = directlock)) &
126             transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
127         THEN
128             xx := dir ||
129             Set(unset, routelock, directlever, dir, dir_l, oppositedirect, existence)
130         END
131     END;
132
133     RouteLock =
134     PRE routelock = unlock & routelever = set & direction = up
135     THEN
136         Set(routelever, lock, directlever, direction, directlock, oppositedirect, existence)
137     END;
138
139     xx <-- RouteUnLock =
140     PRE routelock = lock & direction = up
141     THEN
142         ANY dir, dir_l WHERE
143             dir : DIRECTION & dir_l : LOCK &
144             ((routelever = unset & existence = nondetect) => (dir = directlever & dir_l = unlock)) &
145             ((routelever = set or existence = detect) => (dir = direction & dir_l = directlock)) &
146             transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
147         THEN
148             xx := dir ||
149             Set(routelever, unlock, directlever, dir, dir_l, oppositedirect, existence)
150         END
151     END;
152
153     PSet(r_le, r_lo, d_le, dir, d_lo, op, ex) =
154     PRE
155         r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
156         dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
157         not(dir = up & op = down) &
158         (dir = down => (r_lo = unlock & d_lo = unlock)) &
159         (dir = up =>
160              (((ex = detect or r_lo = lock or r_le = set) => d_lo = lock) &
161               ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock)))
162     THEN
163         Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
164     END
165 END
```

C.2　単線自動閉そく式

注：User Pass を減らすための記述法を適用したものを示す.

C.2.1　AUTOBLOCK マシン（全体定義）

```
MACHINE
    AUTO_BLOCK(nn)

CONSTRAINTS
    nn : NAT1 & nn /= MAXINT

INCLUDES
    OE.OriginEquip,
    DE.DestEquip,
    DC.DirectCircuit,
    UP.TrackC(nn),
    DN.TrackC(nn),
    Const

INVARIANT
    OE.direction = DC.directions(origin) &
    DE.direction = DC.directions(dest) &
    OE.oppositedirect = DC.directions(dest) &
    DE.oppositedirect = DC.directions(origin) &
    OE.existence = DC.exist &
    DE.existence = DC.exist &

    (DC.directions(origin) = up => ran(DN.transmit) = {tstop}) &
    (DC.directions(dest) = down => ran(UP.transmit) = {tstop}) &
    ((DC.directions(origin) = down & DC.directions(dest) = down) =>
        (tstop /: ran(DN.transmit) &
        (DC.exist = detect => detect : ran(DN.existence)) &
        (DC.exist = nondetect => detect /: ran(DN.existence)))) &

    ((DC.directions(origin) = up & DC.directions(dest) = up) =>
        (tstop /: ran(UP.transmit) &
        (DC.exist = detect => detect : ran(UP.existence)) &
        (DC.exist = nondetect => detect /: ran(UP.existence))))

ASSERTIONS
    (DC.directions <+ {origin |-> up})(dest) = DC.directions(dest) &
    (DC.directions <+ {dest |-> down})(origin) = DC.directions(origin)

INITIALISATION
    BEGIN
        DN.PowerOff ||
        UP.PowerOff ||
        DC.Set(up, down, detect) ||
        ANY o_r_le, o_d_le WHERE o_r_le : LEVER & o_d_le : DIRECTION
        THEN
            OE.PSet(o_r_le, unlock, o_d_le, up, unlock, down, detect)
        END ||
        ANY d_r_le, d_d_le WHERE d_r_le : LEVER & d_d_le : DIRECTION
        THEN
            DE.PSet(d_r_le, unlock, d_d_le, down, unlock, up, detect)
        END
    END

OPERATIONS
    xx <-- OriginDownSet =
    BEGIN
        xx <-- OE.DownSet ||
        IF OE.direction = up & OE.oppositedirect = down THEN
            ASSERT DC.directions(origin) = up &
                   DC.directions(dest) = down
            THEN
                DE.OppositeDirectChange(down)||
                DC.DirectionSet(origin, down)||
                DN.Transmit
            END
        END
```

```
67      END;
68
69      xx <-- OriginUpSet =
70      BEGIN
71          xx <-- OE.UpSet ||
72          IF OE.direction = down & OE.directlock = unlock THEN
73              ASSERT DC.directions(origin) = down &
74                     DC.directions(dest) = down &
75                     DE.direction = down &
76                     not((DC.directions <+ {origin |-> up})(dest) = up)
77              THEN
78                  DE.OppositeDirectChange(up) ||
79                  DC.DirectionSet(origin, up) ||
80                  DN.PowerOff
81              END
82          END
83      END;
84
85      xx <-- DestDownSet =
86      BEGIN
87          xx <-- DE.DownSet ||
88          IF DE.direction = up & DE.directlock = unlock THEN
89              ASSERT DC.directions(dest) = up &
90                     DC.directions(origin) = up &
91                     OE.direction = up &
92                     not((DC.directions <+ {dest |-> down})(origin) = down)
93              THEN
94                  OE.OppositeDirectChange(down) ||
95                  DC.DirectionSet(dest, down) ||
96                  UP.PowerOff
97              END
98          END
99      END;
100
101     xx <-- DestUpSet =
102     BEGIN
103         xx <-- DE.UpSet ||
104         IF DE.direction = down & DE.oppositedirect = up THEN
105             ASSERT DC.directions(origin) = up &
106                    DC.directions(dest) = down
107             THEN
108                 OE.OppositeDirectChange(up)||
109                 DC.DirectionSet(dest, up)||
110                 UP.Transmit
111             END
112         END
113     END;
114
115     DTrackCDown(xx) =
116     PRE xx : 1..nn &
117         DN.transmit(xx) /= tstop & DN.existence(xx) = nondetect
118     THEN
119         ASSERT not({tstop} = ran(DN.transmit)) &
120                not(DC.directions(origin) = up) &
121                DC.directions(dest) = down
122         THEN
123             DN.TrackCDown(xx) ||
124             IF DC.exist = nondetect THEN
125                 DC.TrackCDown ||
126                 OE.TrackCDown ||
127                 DE.TrackCDown
128             END
129         END
130     END;
131
132     UTrackCDown(xx) =
133     PRE xx : 1..nn &
134         UP.transmit(xx) /= tstop & UP.existence(xx) = nondetect
135     THEN
136         ASSERT not({tstop} = ran(UP.transmit)) &
137                not(DC.directions(dest) = down) &
138                DC.directions(origin) = up
139         THEN
140             UP.TrackCDown(xx) ||
141             IF DC.exist = nondetect THEN
142                 DC.TrackCDown ||
143                 OE.TrackCDown ||
```

```
144                 DE.TrackCDown
145             END
146         END
147     END;
148
149     DTrackCUp(xx) =
150     PRE xx : 1..nn &
151         DN.transmit(xx) /= tstop & DN.existence(xx) = detect
152     THEN
153         ASSERT not({tstop} = ran(DN.transmit)) &
154                not(DC.directions(origin) = up) &
155                DC.directions(dest) = down &
156                DC.exist = detect
157         THEN
158             IF detect : ran(DN.existence <+ {xx |-> nondetect}) THEN
159                 DN.TrackCUp(xx)
160             ELSE
161                 ASSERT DC.directions(origin) = down &
162                        DC.directions(dest) = down
163                 THEN
164                     OE.TrackCUp ||
165                     IF OE.routelock = unlock & OE.routelever = unset & OE.directlever = up THEN
166                         DC.DirectionSet(origin, up) ||
167                         DN.PowerOff ||
168                         DE.OppositeDirectChange(up)
169                     ELSE
170                         DN.TrackCUp(xx) ||
171                         DC.TrackCUp ||
172                         DE.TrackCUp
173                     END
174                 END
175             END
176         END
177     END;
178
179     UTrackCUp(xx) =
180     PRE
181         xx : 1..nn &
182         UP.transmit(xx) /= tstop & UP.existence(xx) = detect
183     THEN
184         ASSERT not({tstop} = ran(UP.transmit)) &
185                not(DC.directions(dest) = down) &
186                DC.directions(origin) = up &
187                DC.exist = detect
188         THEN
189             IF detect : ran(UP.existence <+ {xx |-> nondetect}) THEN
190                 UP.TrackCUp(xx)
191             ELSE
192                 ASSERT DC.directions(origin) = up &
193                        DC.directions(dest) = up
194                 THEN
195                     DE.TrackCUp ||
196                     IF DE.routelock = unlock & DE.routelever = unset & DE.directlever = down THEN
197                         DC.DirectionSet(dest, down) ||
198                         UP.PowerOff ||
199                         OE.OppositeDirectChange(down)
200                     ELSE
201                         UP.TrackCUp(xx) ||
202                         DC.TrackCUp ||
203                         OE.TrackCUp
204                     END
205                 END
206             END
207         END
208     END;
209
210     OriginRouteSet =
211     PRE OE.routelever = unset
212     THEN
213         OE.RouteSet
214     END;
215
216     xx <-- OriginRouteRelease =
217     PRE OE.routelever = set
218     THEN
219         xx <-- OE.RouteRelease ||
220         IF OE.direction = down & OE.existence = nondetect & OE.routelock = unlock & OE.directlever = up
```

```
221             THEN
222                 ASSERT DE.direction = down &
223                        not((DC.directions <+ {origin |-> up})(dest) = up)
224                 THEN
225                     DC.DirectionSet(origin, up) ||
226                     DE.OppositeDirectChange(up) ||
227                     DN.PowerOff
228                 END
229             END
230         END;
231 
232         DestRouteSet =
233         PRE DE.routelever = unset
234         THEN
235             DE.RouteSet
236         END;
237 
238         xx <-- DestRouteRelease =
239         PRE DE.routelever = set
240         THEN
241             xx <-- DE.RouteRelease ||
242             IF DE.direction = up & DE.existence = nondetect & DE.routelock = unlock & DE.directlever = down
243             THEN
244                 ASSERT OE.direction = up &
245                        not((DC.directions <+ {dest |-> down})(origin) = down)
246                 THEN
247                     DC.DirectionSet(dest, down) ||
248                     OE.OppositeDirectChange(down) ||
249                     UP.PowerOff
250                 END
251             END
252         END;
253 
254         OriginRouteLock =
255         PRE OE.routelock = unlock & OE.routelever = set & OE.direction = down
256         THEN
257             OE.RouteLock
258         END;
259 
260         DestRouteLock =
261         PRE DE.routelock = unlock & DE.routelever = set & DE.direction = up
262         THEN
263             DE.RouteLock
264         END;
265 
266         xx <-- OriginRouteUnLock =
267         PRE OE.routelock = lock & OE.direction = down & xx : DIRECTION
268         THEN
269             xx <-- OE.RouteUnLock ||
270             IF OE.routelever = unset & OE.existence = nondetect & OE.directlever = up THEN
271                 ASSERT DE.direction = down &
272                        not((DC.directions <+ {origin |-> up})(dest) = up)
273                 THEN
274                     DC.DirectionSet(origin, up) ||
275                     DE.OppositeDirectChange(up) ||
276                     DN.PowerOff
277                 END
278             END
279         END;
280 
281         xx <-- DestRouteUnLock =
282         PRE DE.routelock = lock & DE.direction = up & xx : DIRECTION
283         THEN
284             xx <-- DE.RouteUnLock ||
285             IF DE.routelever = unset & DE.existence = nondetect & DE.directlever = down THEN
286                 ASSERT OE.direction = up &
287                        not((DC.directions <+ {dest |-> down})(origin) = down)
288                 THEN
289                     DC.DirectionSet(dest, down) ||
290                     OE.OppositeDirectChange(down) ||
291                     UP.PowerOff
292                 END
293             END
294         END
295 
296 END
```

C.2.2　Const マシン（定数）

```
MACHINE
    Const

SETS
    ASPECT = {green, yellow, red};
    EXIST = {detect, nondetect};
    LOCK = {lock, unlock};
    LEVER = {set, unset};
    DIRECTION = {up, down};
    LOCATION = {origin, dest};
    TRANSMIT = {t90, t45, tstop}

END
```

C.2.3　DirectCircuit マシン（方向回線）

```
MACHINE
    DirectCircuit

SEES
    Const

VARIABLES
    directions

CONCRETE_VARIABLES
    exist

DEFINITIONS
    inv_dc(dir, ex) ==
       dir : LOCATION --> DIRECTION &
       ex : EXIST &
       not(dir(origin) = down & dir(dest) = up) &
       ((dir(origin) = up & dir(dest) = down) => ex = detect)

INVARIANT
    inv_dc(directions, exist)

INITIALISATION
    directions, exist :(inv_dc(directions, exist))

OPERATIONS
    TrackCDown =
    PRE exist = nondetect
    THEN
        exist := detect
    END;

    TrackCUp =
    PRE exist = detect &
        not(directions(origin) = up & directions(dest) = down)
    THEN
        exist := nondetect
    END;

    DirectionSet(loc, dir) =
    PRE loc : LOCATION & dir : DIRECTION &
        not(loc = origin & dir = down & directions(dest) = up) &
        not(loc = dest & dir = up & directions(origin) = down)
    THEN
        directions := directions <+ {loc |-> dir} ||
        IF loc = origin & dir = up & directions(dest) = down
        THEN
            exist := detect
        ELSE IF loc = dest & dir = down & directions(origin) = up
            THEN
                exist := detect
            END
        END
    END;
```

```
55
56      Set(loc1, loc2, ex) =
57      PRE loc1 : DIRECTION & loc2 : DIRECTION & ex : EXIST &
58          not(loc1 = down & loc2 = up) &
59          (loc1 = up & loc2 = down => ex = detect)
60      THEN
61          directions := {origin|-> loc1, dest |-> loc2} ||
62          exist := ex
63      END
64
65  END
```

C.2.4 StationEquip マシン（駅装置）

```
1   MACHINE
2       StationEquip
3
4   SEES
5       Const
6
7   CONCRETE_VARIABLES
8       routelever, routelock,
9       directlever, direction, directlock,
10      oppositedirect, existence
11
12  INVARIANT
13      routelever : LEVER &
14      routelock : LOCK &
15      directlever : DIRECTION &
16      direction : DIRECTION &
17      directlock : LOCK &
18      oppositedirect : DIRECTION &
19      existence : EXIST
20
21  INITIALISATION
22      routelever :: LEVER ||
23      routelock :: LOCK ||
24      directlever :: DIRECTION ||
25      direction :: DIRECTION ||
26      directlock :: LOCK ||
27      oppositedirect :: DIRECTION ||
28      existence :: EXIST
29
30  OPERATIONS
31      Init =
32      BEGIN
33      routelever :: LEVER ||
34      routelock :: LOCK ||
35      directlever :: DIRECTION ||
36      direction :: DIRECTION ||
37      directlock :: LOCK ||
38      oppositedirect :: DIRECTION ||
39      existence :: EXIST
40      END;
41
42      Set(r_le, r_lo, d_le, dir, d_lo, o_dir ,ex) =
43      PRE r_le : LEVER & r_lo : LOCK & d_le : DIRECTION & dir : DIRECTION & d_lo : LOCK &
44          o_dir : DIRECTION & ex : EXIST
45      THEN
46          routelever := r_le ||
47          routelock := r_lo ||
48          directlever := d_le ||
49          direction := dir ||
50          directlock := d_lo ||
51          oppositedirect := o_dir ||
52          existence := ex
53      END
54
55  END
```

C.2.5　OriginEquip マシン（起点方駅装置）

```
MACHINE
    OriginEquip_n

INCLUDES
    StationEquip

SEES
    Const

DEFINITIONS
    transition(dir, d_lo, op, dir2, d_lo2, op2) ==
        ((dir = down & op = down) =>
            (((d_lo = lock & d_lo2 = lock) => dir2 = dir) &
               op2 = op)) &
        ((dir = up & op = up) => dir2 = dir)

INVARIANT
    not(direction = down & oppositedirect = up) &
    (direction = up => (routelock = unlock & directlock = unlock)) &
    (direction = down =>
        ((not(existence = nondetect & routelock = unlock & routelever = unset) => directlock = lock) &
         ((existence = nondetect & routelock = unlock & routelever = unset) => directlock = unlock))) &
    ((oppositedirect = down & direction = up) => existence = detect)

INITIALISATION
    ANY r_le, r_lo, d_le, dir, d_lo, op, ex WHERE
        r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
        dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
        not(dir = down & op = up) &
        (dir = up => (r_lo = unlock & d_lo = unlock)) &
        (dir = down =>
            ((not(ex = nondetect & r_lo = unlock &  r_le = unset) => d_lo = lock) &
             ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))) &
        ((op = down & dir = up) => ex = detect)
    THEN
        Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
    END

OPERATIONS
    xx <-- DownSet =
    IF direction = up & oppositedirect = down THEN
        ASSERT existence = detect &
               not(existence = nondetect & routelock = unlock & routelever = set)
        THEN
            ANY dir, dir_l WHERE
                dir : DIRECTION & dir_l :LOCK &
                dir = down & dir_l = lock &
                transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
            THEN
                Set(routelever, routelock, down, dir, dir_l, oppositedirect ,existence) ||
                xx := dir
            END
        END
    ELSE
        Set(routelever, routelock, down, direction, directlock, oppositedirect ,existence) ||
        xx := direction
    END;

    xx <-- UpSet =
    IF direction = down & directlock = unlock THEN
        ASSERT oppositedirect = down &
               routelock = unlock
        THEN
            ANY dir, ex WHERE
                dir : DIRECTION & ex : EXIST &
                dir = up &  ex = detect &
                transition(direction, directlock, oppositedirect, dir, directlock, oppositedirect)
            THEN
                Set(routelever, routelock, up, dir, directlock, oppositedirect ,ex) ||
                xx := dir
            END
        END
```

```
73      ELSE
74          Set(routelever, routelock, up, direction, directlock, oppositedirect ,existence) ||
75          xx := direction
76      END;
77
78      OppositeDirectChange(o_dir) =
79      PRE o_dir : DIRECTION & direction = up
80      THEN
81          ANY ex WHERE
82              ex : EXIST &
83              (o_dir = down => ex = detect) &
84              (o_dir = up => ex = existence) &
85              transition(direction, directlock, oppositedirect, direction, directlock, o_dir)
86          THEN
87              Set(routelever, routelock, directlever, direction, directlock, o_dir, ex)
88          END
89      END;
90
91      TrackCDown =
92      PRE existence = nondetect
93      THEN
94          IF direction = down THEN
95              ANY ex, dir_l WHERE
96                  ex : EXIST & dir_l : LOCK &
97                  ex = detect & dir_l = lock &
98                  (not(ex = nondetect & routelock = unlock & routelever = unset) => dir_l = lock) &
99                  transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
100             THEN
101                 Set(routelever, routelock, directlever, direction, dir_l, oppositedirect, ex)
102             END
103         ELSE
104             Set(routelever, routelock, directlever, direction, directlock, oppositedirect, detect)
105         END
106     END;
107
108     TrackCUp =
109     PRE existence = detect & not(direction = up & oppositedirect = down)
110     THEN
111         IF direction = down & routelever = unset & routelock = unlock THEN
112             ASSERT oppositedirect = down
113             THEN
114                 ANY dir_l, dir, ex WHERE
115                     dir_l : LOCK & dir : DIRECTION & ex : EXIST &
116                     dir_l = unlock &  dir = directlever &
117                     (dir = up => ex = detect) &
118                     (dir = down => (ex = nondetect & routelock = unlock & routelever = unset)) &
119                     transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
120                 THEN
121                     Set(routelever, routelock, directlever, dir, dir_l, oppositedirect, ex)
122                 END
123             END
124         ELSE
125             ASSERT (direction = down & routelever = unset) => not(routelock = unlock)
126             THEN
127                 Set(routelever, routelock, directlever, direction, directlock, oppositedirect, nondetect)
128             END
129         END
130     END;
131
132     RouteSet =
133     PRE routelever = unset
134     THEN
135         ANY r_le, dir_l WHERE
136             r_le : LEVER & dir_l : LOCK &
137             r_le = set &
138             not(existence = detect & routelock = unlock & r_le = unset) &
139             (direction = down => dir_l = lock) &
140             (direction = up => dir_l = directlock) &
141             transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
142         THEN
143            Set(r_le, routelock, directlever, direction, dir_l, oppositedirect, existence)
144         END
145     END;
146
147     xx <-- RouteRelease =
148     PRE routelever = set
149     THEN
```

```
150          IF direction = down & existence = nondetect & routelock = unlock THEN
151              ASSERT oppositedirect = down
152              THEN
153                  ANY r_le, dir, dir_l, ex WHERE
154                      r_le : LEVER & dir : DIRECTION & dir_l : LOCK & ex : EXIST &
155                      r_le = unset & dir = directlever & dir_l = unlock &
156                      (dir = up => ex = detect) &
157                      (dir = down => ex = nondetect) &
158                      transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
159                  THEN
160                      xx := dir ||
161                      Set(r_le, routelock, directlever, dir, dir_l, oppositedirect, ex)
162                  END
163              END
164           ELSE
165              xx := direction ||
166              Set(unset, routelock, directlever, direction, directlock, oppositedirect, existence)
167          END
168      END;
169
170      RouteLock =
171      PRE routelock = unlock & routelever = set & direction = down
172      THEN
173          Set(routelever, lock, directlever, direction, directlock, oppositedirect, existence)
174      END;
175
176      xx <-- RouteUnLock =
177      PRE routelock = lock & direction = down
178      THEN
179          IF routelever = unset & existence = nondetect THEN
180              ANY dir, dir_l, ex WHERE
181                  dir : DIRECTION & dir_l : LOCK & ex : EXIST &
182                  dir = directlever & dir_l = unlock &
183                  (dir = up => ex = detect) &
184                  (dir = down => ex = nondetect) &
185                  transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
186              THEN
187                  xx := dir ||
188                  Set(routelever, unlock, directlever, dir, dir_l, oppositedirect, ex)
189              END
190          ELSE
191              xx := direction ||
192              Set(routelever, unlock, directlever, direction, directlock, oppositedirect, existence)
193          END
194      END;
195
196      PSet(r_le, r_lo, d_le, dir, d_lo, op, ex) =
197      PRE
198          r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
199          dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
200          not(dir = down & op = up) &
201          (dir = up => (r_lo = unlock & d_lo = unlock)) &
202          (dir = down =>
203              ((not(ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = lock) &
204               ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))) &
205          ((op = down & dir = up) => ex = detect)
206      THEN
207          Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
208      END
209
210  END
```

C.2.6　DestEquip マシン（終点方駅装置）

```
1   MACHINE
2       DestEquip
3
4   INCLUDES
5       StationEquip
6
7   SEES
8       Const
9
10  DEFINITIONS
11      transition(dir, d_lo, op, dir2, d_lo2, op2) ==
```

```
12          ((dir = up & op = up) =>
13              (((d_lo = lock & d_lo2 = lock) => dir2 = dir) &
14               op2 = op)) &
15          ((dir = down & op = down) => dir2 = dir)
16
17  INVARIANT
18      not(direction = up & oppositedirect = down) &
19      (direction = down => (routelock = unlock & directlock = unlock)) &
20      (direction = up =>
21          ((not(existence = nondetect & routelock = unlock & routelever = unset) => directlock = lock) &
22          ((existence = nondetect & routelock = unlock & routelever = unset) => directlock = unlock))) &
23      ((oppositedirect = up & direction = down) => existence = detect)
24
25
26  INITIALISATION
27      ANY r_le, r_lo, d_le, dir, d_lo, op, ex WHERE
28          r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
29          dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
30          not(dir = up & op = down) &
31          (dir = down => (r_lo = unlock & d_lo = unlock)) &
32          (dir = up =>
33              ((not(ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = lock) &
34              ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))) &
35          ((op = up & dir = down) => ex = detect)
36      THEN
37          Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
38      END
39
40  OPERATIONS
41      xx <-- UpSet =
42      IF direction = down & oppositedirect = up THEN
43          ASSERT existence = detect &
44                 not(existence = nondetect & routelock = unlock & routelever = set)
45          THEN
46              ANY dir, dir_l WHERE
47                  dir : DIRECTION & dir_l : LOCK &
48                  dir = up & dir_l = lock &
49                  transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
50              THEN
51                  Set(routelever, routelock, up, dir, dir_l, oppositedirect ,existence) ||
52                  xx := dir
53              END
54          END
55      ELSE
56          Set(routelever, routelock, up, direction, directlock, oppositedirect ,existence) ||
57          xx := direction
58      END;
59
60      xx <-- DownSet =
61      IF direction = up & directlock = unlock THEN
62          ASSERT oppositedirect = up &
63                 routelock = unlock
64          THEN
65              ANY dir, ex WHERE
66                  dir  : DIRECTION & ex : EXIST &
67                  dir = down & ex = detect &
68                  transition(direction, directlock, oppositedirect, dir, directlock, oppositedirect)
69              THEN
70                  Set(routelever, routelock, down, dir, directlock, oppositedirect ,ex) ||
71                  xx := dir
72              END
73          END
74      ELSE
75          Set(routelever, routelock, down, direction, directlock, oppositedirect ,existence) ||
76          xx := direction
77      END
78      ;
79
80      OppositeDirectChange(o_dir) =
81      PRE o_dir : DIRECTION & direction = down
82      THEN
83          ANY ex WHERE
84              ex : EXIST &
85              (o_dir = up => ex = detect) &
86              (o_dir = down => ex = existence) &
87              transition(direction, directlock, oppositedirect, direction, directlock, o_dir)
88          THEN
```

```
            Set(routelever, routelock, directlever, direction, directlock, o_dir ,ex)
        END
    END;

    TrackCDown =
    PRE existence = nondetect
    THEN
        IF direction = up THEN
            ANY ex, dir_l WHERE ex : EXIST & dir_l : LOCK &
                ex = detect & dir_l = lock &
                (not(ex = nondetect & routelock = unlock & routelever = unset) => dir_l = lock) &
                transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
            THEN
                Set(routelever, routelock, directlever, direction, dir_l, oppositedirect, ex)
            END
        ELSE
            Set(routelever, routelock, directlever, direction, directlock, oppositedirect, detect)
        END
    END;

    TrackCUp =
    PRE existence = detect &
        not(direction = down & oppositedirect = up)
    THEN
        IF direction = up & routelever = unset & routelock = unlock THEN
            ASSERT oppositedirect = up
            THEN
                ANY dir_l, dir, ex WHERE
                    dir_l : LOCK & dir : DIRECTION & ex : EXIST &
                    dir_l = unlock &  dir = directlever &
                    (dir = down => ex = detect) &
                    (dir = up => (ex = nondetect & routelock = unlock & routelever = unset)) &
                    transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
                THEN
                    Set(routelever, routelock, directlever, dir, dir_l, oppositedirect, ex)
                END
            END
        ELSE
            ASSERT (direction = up & routelever = unset) => not(routelock = unlock)
            THEN
                Set(routelever, routelock, directlever, direction, directlock, oppositedirect, nondetect)
            END
        END
    END;

    RouteSet =
    PRE routelever = unset
    THEN
        ANY r_le, dir_l WHERE
            r_le : LEVER & dir_l : LOCK &
            r_le = set &
            not(existence = detect & routelock = unlock & r_le = unset) &
            (direction = up => dir_l = lock) &
            (direction = down => dir_l = directlock) &
            transition(direction, directlock, oppositedirect, direction, dir_l, oppositedirect)
        THEN
           Set(r_le, routelock, directlever, direction, dir_l, oppositedirect, existence)
        END
    END;

    xx <-- RouteRelease =
    PRE routelever = set
    THEN
        IF direction = up & existence = nondetect & routelock = unlock THEN
            ASSERT oppositedirect = up
            THEN
                ANY r_le, dir, dir_l, ex  WHERE
                    r_le : LEVER & dir : DIRECTION & dir_l : LOCK & ex : EXIST &
                    r_le = unset & dir = directlever & dir_l = unlock &
                    (dir = down => ex = detect) &
                    (dir = up => ex = nondetect) &
                    transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
                THEN
                    xx := dir ||
                    Set(r_le, routelock, directlever, dir, dir_l, oppositedirect, ex)
                END
            END
```

```
166             ELSE
167                 xx := direction ||
168                 Set(unset, routelock, directlever, direction, directlock, oppositedirect, existence)
169             END
170         END;
171
172         RouteLock =
173         PRE routelock = unlock & routelever = set & direction = up
174         THEN
175             Set(routelever, lock, directlever, direction, directlock, oppositedirect, existence)
176         END;
177
178         xx <-- RouteUnLock =
179         PRE routelock = lock & direction = up
180         THEN
181             IF routelever = unset & existence = nondetect THEN
182                 ANY dir, dir_l, ex WHERE
183                     dir : DIRECTION & dir_l : LOCK & ex : EXIST &
184                     dir = directlever & dir_l = unlock &
185                     (dir = down => ex = detect) &
186                     (dir = up => ex = nondetect) &
187                     transition(direction, directlock, oppositedirect, dir, dir_l, oppositedirect)
188                 THEN
189                     xx := dir ||
190                     Set(routelever, unlock, directlever, dir, dir_l, oppositedirect, ex)
191                 END
192             ELSE
193                 xx := direction ||
194                 Set(routelever, unlock, directlever, direction, directlock, oppositedirect, existence)
195             END
196         END;
197
198         PSet(r_le, r_lo, d_le, dir, d_lo, op, ex) =
199         PRE
200             r_le : LEVER & r_lo : LOCK & d_le :DIRECTION &
201             dir : DIRECTION & d_lo : LOCK & op : DIRECTION & ex : EXIST &
202             not(dir = up & op = down) &
203             (dir = down => (r_lo = unlock & d_lo = unlock)) &
204             (dir = up =>
205                 ((not(ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = lock) &
206                 ((ex = nondetect & r_lo = unlock & r_le = unset) => d_lo = unlock))) &
207                 ((op = up & dir = down) => ex = detect)
208         THEN
209             Set(r_le, r_lo, d_le, dir, d_lo, op ,ex)
210         END
211
212     END
```

C.2.7 TrackC マシン（軌道回路）

```
1   MACHINE
2       TrackC(nn)
3
4   SEES
5       Const
6
7   CONSTRAINTS
8       nn : NAT1 & nn /= MAXINT
9
10  DEFINITIONS
11      TC == (1..nn);
12
13      inv_tc(as, tr, ex) == (
14
15      as : TC --> ASPECT &
16      tr : TC --> TRANSMIT &
17      ex : TC --> EXIST &
18
19      (tstop : ran(tr) => {tstop} = ran(tr)) &
20      !ii.(ii : TC => (
21          (ex(ii) = detect => as(ii) = red) &
22          (tr(ii) = tstop => ex(ii) = detect) &
23          ((tr(ii) = t90 & ex(ii) = nondetect) => as(ii) = green) &
24          ((tr(ii) = t45 & ex(ii) = nondetect) => as(ii) = yellow))) &
25      !ii.(ii : TC => ((ii + 1 : TC & tr(ii) /= tstop) =>
```

```
26          ((tr(ii + 1) = t45 => as(ii) = red) &
27          (as(ii) = red => tr(ii + 1) = t45)))))
28
29  VARIABLES
30      aspect, transmit, existence
31
32  INVARIANT
33      inv_tc(aspect, transmit, existence)
34
35  INITIALISATION
36      aspect, transmit, existence :(inv_tc(aspect, transmit, existence))
37
38  OPERATIONS
39      Transmit =
40      PRE {tstop} = ran(transmit)
41      THEN
42          ANY as, tr, ex WHERE
43              inv_tc(as, tr, ex) &
44              tstop /: ran(tr) &
45              ex = TC * {detect} &
46              as = TC * {red}
47          THEN
48              aspect := as ||
49              transmit := tr ||
50              existence := ex
51          END
52      END;
53
54      PowerOff =
55      BEGIN
56          aspect := TC * {red} ||
57          transmit := TC * {tstop} ||
58          existence := TC * {detect}
59      END;
60
61      TrackCDown(xx) =
62      PRE xx : TC &
63          transmit(xx) /= tstop &
64          existence(xx) = nondetect
65      THEN
66          ANY as, tr, ex WHERE
67              inv_tc(as, tr, ex) &
68              ex = existence <+ {xx |-> detect} &
69              (not(tr = transmit) => tr = transmit <+ {xx +1 |-> t45}) &
70              (xx + 1 : TC & transmit(xx + 1) /= tstop => tr = transmit <+ {xx + 1 |-> t45}) &
71              (not(xx + 1 : TC & transmit(xx + 1) /= tstop) => tr = transmit)
72          THEN
73              aspect := as ||
74              transmit := tr ||
75              existence := ex
76          END
77      END;
78
79      TrackCUp(xx) =
80      PRE xx : TC &
81          transmit(xx) /= tstop &
82          existence(xx) = detect
83      THEN
84          ANY as, tr, ex WHERE
85              inv_tc(as, tr, ex) &
86              ex = existence <+ {xx |-> nondetect} &
87              (not(tr = transmit) => tr = transmit <+ {xx +1 |-> t90}) &
88              (xx + 1 : TC & transmit(xx + 1) /= tstop => tr = transmit <+ {xx + 1 |-> t90}) &
89              (not(xx + 1 /: TC & transmit(xx + 1) /= tstop) => tr = transmit)
90          THEN
91              aspect := as ||
92              transmit := tr ||
93              existence := ex
94          END
95      END;
96
97      xx <-- Existence =
98      BEGIN
99          IF detect : ran(existence) THEN xx := detect ELSE xx := nondetect END
100     END;
101
102     ex <-- Exist(xx) =
```

```
103       PRE xx : TC
104       THEN ex := existence(xx)
105       END;
106
107       tr <-- Trans(xx) =
108       PRE xx : TC
109       THEN tr := transmit(xx)
110       END
111
112   END
```

C.3 特殊自動閉そく式

注：User Pass を減らすための記述法を適用したものを示す.

C.3.1 SEMIAUTOBLOCK マシン（全体定義）

```
1   MACHINE
2       SEMIAUTOBLOCK
3
4   INCLUDES
5       Const,
6       OE.Station(0),
7       DE.Station(1),
8       DC.DirectCircuit
9
10  INVARIANT
11      OE.dir = DC.directions(origin) &
12      OE.mis = DC.misdep(origin) &
13      (OE.dir = down => DC.arrive(origin) = nondetect) &
14      ((OE.dir = up & OE.pos = ct & OE.mis = nondetect) => not(DC.arrive(origin) = nondetect)) &
15      (not(DC.arrive(origin) = nondetect) => (OE.dir = up & OE.pos = ct & OE.mis = nondetect)) &
16      OE.o_dir = DC.directions(dest) &
17      OE.o_arr = DC.arrive(dest) &
18      OE.o_mis = DC.misdep(dest) &
19
20      DE.dir = DC.directions(dest) &
21      DE.mis = DC.misdep(dest) &
22      (DE.dir = up => DC.arrive(dest) = nondetect) &
23      ((DE.dir = down & DE.pos = ct & DE.mis = nondetect) => not(DC.arrive(dest) = nondetect)) &
24      (not(DC.arrive(dest) = nondetect) => (DE.dir = down & DE.pos = ct & DE.mis = nondetect)) &
25      DE.o_dir = DC.directions(origin) &
26      DE.o_arr = DC.arrive(origin) &
27      DE.o_mis = DC.misdep(origin)
28
29  INITIALISATION
30      OE.SetParam(unset, unlock, red, unset, unset, up, up, unlock, down, nondetect, nondetect, between,
31                  nondetect, nondetect, nondetect, nondetect, nondetect, nondetect) ||
32      DE.SetParam(unset, unlock, red, unset, unset, down, down, unlock, up, nondetect, nondetect, between,
33                  nondetect, nondetect, nondetect, nondetect, nondetect, nondetect) ||
34      DC.Set(up, down, nondetect, nondetect, nondetect, nondetect)
35
36  ASSERTIONS
37      ((DC.directions(origin) = up & DC.directions(dest) = down) =>
38          (OE.pos = between & DE.pos = between)) &
39      ((DC.directions(origin) = up & DC.directions(dest) = down) =>
40          (DC.arrive(origin) = nondetect & DC.arrive(dest) = nondetect))
41
42  OPERATIONS
43      OriginDownSET =
44      PRE OE.d_lev = up
45      THEN
46          OE.SetDepart ||
47          IF OE.dir = up & OE.o_dir = down & OE.o_mis = nondetect & OE.mis = nondetect THEN
48              ASSERT (DC.directions <+ {origin |-> down})(dest) = DC.directions(dest) &
49                     DC.directions(origin) = up &
50                     DC.directions(dest) = down
51              THEN
52                  DE.OppositeChange(down) ||
53                  DC.DirectionSet(origin, down)
54              END
55          END
56      END;
```

```
57
58  OriginUpSET =
59      PRE OE.d_lev = down
60      THEN
61          OE.SetArrive ||
62          IF OE.dir = down & OE.d_lock = unlock & OE.o_mis = nondetect & OE.o_arr = nondetect THEN
63              ASSERT DC.directions(origin) = down &
64                     DC.directions(dest) = down &
65                     DC.arrive(dest) = nondetect &
66                     not(DC.arrive(origin) = detect)
67              THEN
68                  DE.OppositeChange(up) ||
69                  DC.DirectionSet(origin, up)
70              END
71          END
72      END;
73
74      OriginDirectionUnlocked =
75      PRE OE.dir = down & OE.d_lev = up & OE.d_lock = unlock & OE.o_mis = nondetect & OE.o_arr = nondetect
76      THEN
77          ASSERT DC.directions(origin) = down &
78                 DC.directions(dest) = down &
79                 DC.arrive(dest) = nondetect &
80                 not(DC.arrive(origin) = detect)
81          THEN
82              OE.SetArrive ||
83              DC.DirectionSet(origin, up) ||
84              DE.OppositeChange(up)
85          END
86      END;
87
88      DestDirectionUnlocked =
89      PRE DE.dir = up & DE.d_lev = down & DE.d_lock = unlock & DE.o_mis = nondetect & DE.o_arr = nondetect
90      THEN
91          ASSERT DC.directions(dest) = up &
92                 DC.directions(origin) = up &
93                 DC.arrive(origin) = nondetect &
94                 not(DC.arrive(dest) = detect)
95          THEN
96              DE.SetArrive ||
97              DC.DirectionSet(dest, down) ||
98              OE.OppositeChange(down)
99          END
100     END;
101
102     DestDownSET =
103     PRE DE.d_lev = up
104     THEN
105         DE.SetArrive ||
106         IF DE.dir = up & DE.d_lock = unlock & DE.o_mis = nondetect & DE.o_arr = nondetect THEN
107             ASSERT DC.directions(dest) = up &
108                    DC.directions(origin) = up &
109                    DC.arrive(origin) = nondetect &
110                    not(DC.arrive(dest) = detect)
111             THEN
112                 OE.OppositeChange(down) ||
113                 DC.DirectionSet(dest, down)
114             END
115         END
116     END;
117
118     DestUpSET =
119     PRE DE.d_lev = down THEN
120         DE.SetDepart ||
121         IF DE.dir = down & DE.o_dir = up & DE.o_mis = nondetect & DE.mis = nondetect THEN
122             ASSERT (DC.directions <+ {dest |-> up})(origin) = DC.directions(origin) &
123                    DC.directions(origin) = up &
124                    DC.directions(dest) = down
125             THEN
126                 OE.OppositeChange(up) ||
127                 DC.DirectionSet(dest, up)
128             END
129         END
130     END;
131
132     OriginRouteSet =
133     PRE OE.r_lev = unset
```

```
134       THEN
135           OE.DepartureRouteSet
136       END;
137
138       DestRouteSet =
139       PRE DE.r_lev = unset
140       THEN
141           DE.DepartureRouteSet
142       END;
143
144       OriginRouteRelease =
145       PRE OE.r_lev = set
146       THEN
147           OE.DepartureRouteRelease
148       END;
149
150       DestRouteRelease =
151       PRE DE.r_lev = set
152       THEN
153           DE.DepartureRouteRelease
154       END;
155
156       OriginHomeTrackDetect =
157       PRE OE.tc(home) = nondetect
158       THEN
159           OE.HomeTrackDetect
160       END;
161
162       OriginHomeTrackNondetect =
163       PRE OE.tc(home) = detect
164       THEN
165           OE.HomeTrackNondetect
166       END;
167
168       DestHomeTrackDetect =
169       PRE DE.tc(home) = nondetect
170       THEN
171           DE.HomeTrackDetect
172       END;
173
174       DestHomeTrackNondetect =
175       PRE DE.tc(home) = detect
176       THEN
177           DE.HomeTrackNondetect
178       END;
179
180       OriginRouteTrackDetect =
181       PRE OE.tc(route) = nondetect THEN
182           OE.RouteTrackDetect
183       END;
184
185       DestRouteTrackDetect =
186       PRE DE.tc(route) = nondetect THEN
187           DE.RouteTrackDetect
188       END;
189
190       OriginRouteTrackNondetect =
191       PRE OE.tc(route) = detect THEN
192           OE.RouteTrackNondetect
193       END;
194
195       DestRouteTrackNondetect =
196       PRE DE.tc(route) = detect THEN
197           DE.RouteTrackNondetect
198       END;
199
200       OriginCTDetect =
201       PRE OE.tc(ct) = nondetect
202       THEN
203           ASSERT (DC.misdep <+ {origin |-> detect})(dest) = DC.misdep(dest) &
204                  (DC.arrive <+ {origin |-> nondetect})(dest) = DC.arrive(dest) &
205                  (not((DC.arrive <+ {origin |-> nondetect})(dest) = nondetect) =>
206                      (DE.dir = down & DE.pos = ct & DE.mis = nondetect))
207           THEN
208               OE.CTDetect ||
209               IF OE.dir = down & OE.mis = nondetect THEN
210                   IF OE.tc(route) = detect & (OE.s_relay = unset => (OE.pos /= route & OE.pos /= ct)) THEN
```

```
                    ASSERT DE.dir = down
                    THEN
                        DC.MisDeparture(origin) ||
                        DE.OppositeMisDeparture
                    END
                END
            ELSIF OE.dir = up & OE.tc(route) = detect THEN
                IF OE.mis = nondetect THEN
                    DC.MisDeparture(origin) ||
                    DE.OppositeMisDeparture
                END
            END
        END;

    DestCTDetect =
    PRE DE.tc(ct) = nondetect
    THEN
        ASSERT (DC.misdep <+ {dest |-> detect})(origin) = DC.misdep(origin) &
               (DC.arrive <+ {dest |-> nondetect})(origin) = DC.arrive(origin) &
               (not((DC.arrive <+ {dest |-> nondetect})(orgin) = nondetect) =>
                   (OE.dir = up & OE.pos = ct & OE.mis = nondetect))
        THEN
            DE.CTDetect ||
            IF DE.dir = up & DE.mis = nondetect THEN
                IF DE.tc(route) = detect & (DE.s_relay = unset => DE.pos /: route & DE.pos /= ct) THEN
                    ASSERT OE.dir = up
                    THEN
                        DC.MisDeparture(dest) ||
                        OE.OppositeMisDeparture
                    END
                END
            ELSIF DE.dir = down & DE.tc(route) = detect THEN
                IF DE.mis = nondetect THEN
                    DC.MisDeparture(dest) ||
                    OE.OppositeMisDeparture
                END
            END
        END
    END;

    OriginCTNondetect =
    PRE OE.tc(ct) = detect
    THEN
        ASSERT (DC.misdep <+ {origin |-> nondetect})(dest) = DC.misdep(dest) &
               (DC.arrive <+ {origin |-> nondetect})(dest) = DC.arrive(dest) &
               (not((DC.arrive <+ {origin |-> nondetect})(dest) = nondetect) =>
                    (DE.dir = down & DE.pos = ct & DE.mis = nondetect))
        THEN
            OE.CTNondetect ||
            IF OE.dir = up & OE.pos = ct & OE.mis = nondetect THEN
                DC.Arrive(origin) ||
                DE.OppositeArrive
            END
        END
    END;

    DestCTNondetect =
    PRE DE.tc(ct) = detect
    THEN
        ASSERT (DC.misdep <+ {dest |-> nondetect})(origin) = DC.misdep(origin) &
               (DC.arrive <+ {dest |-> nondetect})(origin) = DC.arrive(origin) &
               (not((DC.arrive <+ {dest |-> nondetect})(origin) = nondetect) =>
                    (OE.dir = up & OE.pos = ct & OE.mis = nondetect))
        THEN
            DE.CTNondetect ||
            IF DE.dir = down & DE.pos = ct & DE.mis = nondetect THEN
                DC.Arrive(dest) ||
                OE.OppositeArrive
            END
        END
    END;

    OriginOTDetect =
    PRE OE.tc(ot) = nondetect
    THEN
        OE.OTDetect
```

```
288     END;
289
290     DestOTDetect =
291     PRE DE.tc(ot) = nondetect
292     THEN
293         DE.OTDetect
294     END;
295
296     OriginOTNondetect =
297     PRE OE.tc(ot) = detect
298     THEN
299         ASSERT (DC.arrive <+ {origin |-> detect})(dest) = DC.arrive(dest) &
300                (not((DC.arrive <+ {origin |-> detect})(dest) = nondetect) =>
301                (DE.dir = down & DE.pos = ct & DE.mis = nondetect))
302         THEN
303             OE.OTNondetect ||
304             IF OE.dir = up & OE.pos = otct & OE.mis = nondetect THEN
305                 DC.PrepareArrival(origin) ||
306                 DE.OppositeArrivePrepare
307             END
308         END
309     END;
310
311     DestOTNondetect =
312     PRE DE.tc(ot) = detect
313     THEN
314         DE.OTNondetect ||
315         IF DE.dir = down & DE.pos = otct & DE.mis = nondetect THEN
316             DC.PrepareArrival(dest) ||
317             OE.OppositeArrivePrepare
318         END
319     END;
320
321     OriginSubLeverSet =
322     PRE OE.s_lev = unset
323     THEN
324         OE.SubLeverSet
325     END;
326
327     OriginSubLeverRelease =
328     PRE OE.s_lev = set
329     THEN
330         OE.SubLeverRelease
331     END;
332
333     DestSubLeverSet =
334     PRE DE.s_lev = unset
335     THEN
336         DE.SubLeverSet
337     END;
338
339
340     DestSubLeverRelease =
341     PRE DE.s_lev = set
342     THEN
343         DE.SubLeverRelease
344     END;
345
346     OriginBackwardRelease =
347     BEGIN
348         OE.BackwardRelease
349     END;
350
351     DestBackwardRelease =
352     BEGIN
353         DE.BackwardRelease
354     END;
355
356     OriginMisDepartureRelease =
357     BEGIN
358         OE.MisDepartureRelease ||
359         IF OE.mis = detect THEN
360             IF OE.dir = down THEN
361                 IF OE.pos = ct & OE.tc(ct) = nondetect & OE.tc(ot) = nondetect & OE.tc(route) = nondetect
362                 THEN
363                     DC.MisDepReset(origin) ||
364                     DE.OppositeReleaseMisdeparture
```

```
                ELSIF OE.pos : {ot, between, arrival} & OE.tc(ct) = nondetect & OE.tc(route) = nondetect
                THEN
                    DC.MisDepReset(origin) ||
                    DE.OppositeReleaseMisdeparture
                END
            ELSIF OE.tc(ct) = nondetect & OE.tc(route) = nondetect THEN
                DC.MisDepReset(origin) ||
                DE.OppositeReleaseMisdeparture
            END
        END
    END

    DestisDepartureRelease =
    BEGIN
        DE.MisDepartureRelease ||
        IF DE.mis = detect THEN
            IF DE.dir = up THEN
                IF DE.pos = ct & DE.tc(ct) = nondetect & DE.tc(ot) = nondetect & DE.tc(route) = nondetect
                THEN
                    DC.MisDepReset(dest) ||
                    OE.OppositeReleaseMisdeparture
                ELSIF DE.pos : {ot, between, arrival} & DE.tc(ct) = nondetect & DE.tc(route) = nondetect
                THEN
                    DC.MisDepReset(dest) ||
                    OE.OppositeReleaseMisdeparture
                END
            ELSIF DE.tc(ct) = nondetect & DE.tc(route) = nondetect THEN
                DC.MisDepReset(dest) ||
                OE.OppositeReleaseMisdeparture
            END
        END
    END

END
```

C.3.2　CONST マシン（定数）

```
MACHINE
    Const

SETS
    ASPECT = {green, red};
    LOCK = {lock, unlock};
    LEVER = {set, unset};
    DIRECTION = {up, down};
    LOCATION = {origin, dest};
    DETECT = {detect, nondetect};
    POSITION = {inexistent, home, route, ct, otct, ot, between, approach, arrival}

END
```

C.3.3　DirectCircuit マシン（方向回線）

```
MACHINE
    DirectCircuit

SEES
    Const

VARIABLES
    directions, arrive, misdep

INVARIANT
    directions : LOCATION --> DIRECTION &
    arrive : LOCATION --> DETECT &
    misdep : LOCATION --> DETECT &

    not (directions(origin) = down & directions(dest) = up) &
    (arrive(dest) = detect => (directions(origin) = down & directions(dest) = down)) &
    (arrive(origin) = detect => (directions(origin) = up & directions(dest) = up)) &
    not(arrive(dest) = detect & arrive(origin) = detect)

INITIALISATION
```

```
21      misdep :: LOCATION --> DETECT ||
22      directions, arrive :(
23          directions : LOCATION --> DIRECTION &
24          arrive : LOCATION --> DETECT &
25
26          not (directions(origin) = down & directions(dest) = up) &
27          (arrive(dest) = detect => (directions(origin) = down & directions(dest) = down)) &
28          (arrive(origin) = detect => (directions(origin) = up & directions(dest) = up ))&
29          not(arrive(dest) = detect & arrive(origin) = detect)
30      )
31
32  OPERATIONS
33      DirectionSet(loc, dir) =
34      PRE
35          loc : LOCATION & dir : DIRECTION
36      THEN
37          IF
38              not(loc = origin & dir = down & directions(dest) = up) &
39              not(loc = dest & dir = up & directions(origin) = down) &
40              arrive(dest) = nondetect & arrive(origin) = nondetect
41          THEN
42              directions := directions <+ {loc |-> dir}
43          ELSE
44              skip
45          END
46      END;
47
48      PrepareArrival(loc) =
49      PRE
50          loc : LOCATION &
51          arrive(loc) = nondetect &
52          (loc = dest => (directions(origin) = down & directions(dest) = down)) &
53          (loc = origin => (directions(origin) = up & directions(dest) = up))
54      THEN
55          arrive := arrive <+ {loc |-> detect}
56      END;
57
58      Arrive(loc) =
59      PRE
60          loc : LOCATION &
61          arrive(loc) = detect
62      THEN
63          arrive := arrive <+ {loc |-> nondetect}
64      END;
65
66      MisDeparture(loc) =
67      PRE
68          loc : LOCATION &
69          misdep(loc) = nondetect
70      THEN
71          misdep := misdep <+ {loc |-> detect} ||
72          arrive := arrive <+ {loc |-> nondetect}
73      END;
74
75      MisDepReset(loc) =
76      PRE
77          loc : LOCATION &
78          misdep(loc) = detect
79      THEN
80          misdep := misdep <+ {loc |-> nondetect}
81      END;
82
83      Set(loc1, loc2, arr1, arr2, mis1, mis2) =
84      PRE
85          loc1 : DIRECTION & loc2 : DIRECTION & arr1 : DETECT & arr2 : DETECT &
86          mis1 : DETECT & mis2 : DETECT &
87          not (loc1 = down & loc2 = up) &
88          (arr2 = detect => (loc1 = down & loc2 = down)) &
89          (arr1 = detect => (loc1 = up & loc2 = up)) &
90          not(arr1 = detect & arr2 = detect)
91      THEN
92          directions := {origin |-> loc1, dest |-> loc2} ||
93          arrive := {origin |-> arr1, dest |-> arr2} ||
94          misdep := {origin |-> mis1, dest |-> mis2}
95      END
96
97  END
```

C.3.4 Station マシン（駅装置）

```
MACHINE
    Station(Loc)

CONSTRAINTS
    Loc : NAT & (Loc = 0 or Loc = 1)

SEES
    Const

VARIABLES
    dep, arr, r_lev, r_lock, r_aspect, s_lev, s_relay,
    d_lev, dir, d_lock, o_dir, o_arr, o_mis, pos, tc, mis, app

INVARIANT
    dep : DIRECTION & arr : DIRECTION & r_lev : LEVER & r_lock : LOCK & r_aspect : ASPECT &
    s_lev : LEVER & s_relay : LEVER & d_lev : DIRECTION & dir : DIRECTION & d_lock : LOCK &
    o_dir : DIRECTION & o_arr : DETECT & o_mis : DETECT &  pos : POSITION & tc : POSITION +-> DETECT &
    mis : DETECT & app : DETECT &
    (Loc = 0 => (dep = down & arr = up)) &
    (Loc = 1 => (dep = up & arr = down)) &
    dom(tc) = {home, route, ct, ot} &

    (dir = dep =>
         (pos : {inexistent, home, route, ct, otct, ot, between, arrival} &
         (pos = arrival => (not(o_arr = nondetect) & o_mis = nondetect)) &
         (pos : {ct, otct, ot, between, arrival} => d_lock = lock) &
         (pos = home => tc(home) = detect) &
         (pos = inexistent => tc(home) = nondetect) &
         (pos = otct => (s_relay = unset => (tc(ot) = detect & tc(ct) = detect))) &
         (pos = otct => mis = nondetect) &
         (pos = route => (r_lock = lock & tc(route) = detect & tc(ct) = nondetect)) &
         (pos /: {ct, otct, ot, between, arrival} => mis = nondetect))) &

         (r_lock = lock => dir = dep) &
         (r_lock = lock => d_lock = lock) &
         (r_lock = lock => s_relay = unset) &
         ((r_lev = unset & tc(route) = nondetect & tc(ct) = nondetect) => r_lock = unlock) &
         ((dir = dep & r_lev = set & s_relay = unset) => r_lock = lock) &

         (d_lock = lock => dir = dep) &
         (dir = dep & o_mis = detect => d_lock = lock) &
         ((r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect) =>
              d_lock = unlock) &
        (dir = dep => o_dir = dep) &
        (dir = arr =>
             (pos : {between, approach, ot, otct, ct} &
             (o_dir = dep => pos = between))) &

    (r_aspect = green =>
        (r_lev = set & r_lock = lock & s_relay = unset &
        (not(pos = home) => pos = inexistent) &
        tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect &
            mis = nondetect & o_mis = nondetect)) &
    (s_lev = unset => s_relay = unset) &
    (s_lev = set & r_lock = unlock & dir = arr => s_relay = set) &
    (s_lev = set & r_lock = unlock & r_lev = unset => s_relay = set)

ASSERTIONS
    dep /= arr &
    (pos : {ct, otct, ot, between, arrival} =>
    pos : {inexistent, home, route, ct, otct, ot, between, arrival})

INITIALISATION
    dep, arr, r_lev, r_lock, r_aspect, s_lev, s_relay,
    d_lev, dir, d_lock, o_dir, o_arr, o_mis, pos, tc, mis, app :(
    dep : DIRECTION & arr : DIRECTION & r_lev : LEVER & r_lock : LOCK & r_aspect : ASPECT &
    s_lev : LEVER & s_relay : LEVER & d_lev : DIRECTION & dir : DIRECTION & d_lock : LOCK &
    o_dir : DIRECTION & o_arr : DETECT & o_mis : DETECT & pos : POSITION & tc : POSITION +-> DETECT &
    mis : DETECT & app : DETECT &
    (Loc = 0 => (dep = down & arr = up)) &
    (Loc = 1 => (dep = up & arr = down)) &
    dom(tc) = {home, route, ct, ot} &
```

```
73
74      (dir = dep =>
75          (pos : {inexistent, home, route, ct, otct, ot, between, arrival} &
76          (pos = arrival => (not(o_arr = nondetect) & o_mis = nondetect)) &
77          (pos : {ct, otct, ot, between, arrival} => d_lock = lock) &
78          (pos = home => tc(home) = detect) &
79          (pos = inexistent => tc(home) = nondetect) &
80          (pos = otct => (s_relay = unset => (tc(ot) = detect & tc(ct) = detect))) &
81          (pos = otct => mis = nondetect) &
82          (pos = route =>
83               (r_lock = lock & tc(route) = detect & tc(ct) = nondetect)) &
84               (pos /: {ct, otct, ot, between, arrival} => mis = nondetect))) &
85
86          (r_lock = lock => dir = dep) &
87          (r_lock = lock => d_lock = lock) &
88          (r_lock = lock => s_relay = unset) &
89          ((r_lev = unset & tc(route) = nondetect & tc(ct) = nondetect) => r_lock = unlock) &
90          ((dir = dep & r_lev = set & s_relay = unset) => r_lock = lock) &
91
92          (d_lock = lock => dir = dep) &
93          (dir = dep & o_mis = detect => d_lock = lock) &
94          ((r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect) =>
95               d_lock = unlock) &
96          (dir = dep => o_dir = dep) &
97          (dir = arr =>
98              (pos : {between, approach, ot, otct, ct} &
99              (o_dir = dep => pos = between))) &
100
101         (r_aspect = green =>
102             (r_lev = set & r_lock = lock & s_relay = unset &
103             (not(pos = home) => pos = inexistent) &
104             tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect &
105             mis = nondetect & o_mis = nondetect)) &
106
107         (s_lev = unset => s_relay = unset) &
108         (s_lev = set & r_lock = unlock & dir = arr => s_relay = set) &
109         (s_lev = set & r_lock = unlock & r_lev = unset => s_relay = set)
110     )
111
112 OPERATIONS
113     SetDepart =
114     BEGIN
115         d_lev := dep  ||
116         IF dir = arr & o_dir = dep & o_mis = nondetect & mis = nondetect THEN
117             ASSERT r_lock = unlock & d_lock = unlock
118             THEN
119                 dir := dep ||
120                 IF r_lev = set & s_relay = unset THEN
121                     r_lock := lock ||
122                     d_lock := lock ||
123                     IF tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect THEN
124                         r_aspect := green
125                     END
126                 END ||
127                 pos :(pos : POSITION &
128                     pos /:{ct, otct, ot, between, arrival} &
129                     (not(pos = home) => pos = inexistent) &
130                     (pos = home => tc(home) = detect) &
131                     (pos = inexistent => tc(home) = nondetect))
132             END
133         END
134     END;
135
136     SetArrive =
137     BEGIN
138         d_lev := arr ||
139         IF dir = dep & d_lock = unlock & o_mis = nondetect & o_arr = nondetect THEN
140             ASSERT not(r_lock = lock)
141             THEN
142                 dir := arr ||
143                 pos := between ||
144                 IF s_lev = set THEN s_relay := set END
145             END
146         END
147     END;
148
149     OppositeChange(dd) =
```

```
150     PRE
151         dd : DIRECTION &
152         dir = arr & o_arr = nondetect &
153         (dd = arr => (mis = nondetect & o_mis = nondetect))
154     THEN
155         o_dir := dd ||
156         pos := between
157     END;
158
159     OppositeArrivePrepare =
160     PRE dir = dep & o_arr = nondetect & o_mis = nondetect
161     THEN
162         o_arr := detect ||
163         IF pos = between THEN
164             ASSERT pos :{ct, otct, ot, between, arrival} &
165                    not(r_aspect = green)
166             THEN
167                 pos := arrival
168             END
169         END
170     END;
171
172     OppositeArrive =
173     PRE dir = dep & not(o_arr = nondetect) & o_mis = nondetect
174     THEN
175         o_arr := nondetect ||
176         IF pos = arrival THEN
177             ASSERT pos : {ct, otct, ot, between, arrival} &
178                    not(r_aspect = green)
179             THEN
180                 IF mis = detect THEN pos := ct
181                 ELSIF r_lock = lock    THEN
182                     pos :(pos : POSITION &
183                     pos /:{ct, otct, ot, between, arrival} &
184                     (not(pos = home) => pos = inexistent) &
185                     (pos = home => tc(home) = detect) &
186                     (pos = inexistent => tc(home) = nondetect)) ||
187                     IF r_lev = set & tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect THEN
188                         r_aspect := green
189                     END
190             ELSE
191                 d_lock := unlock ||
192                 pos :(pos : POSITION &
193                       pos /:{ct, otct, ot, between, arrival} &
194                       (not(pos = home) => pos = inexistent) &
195                       (pos = home => tc(home) = detect) &
196                       (pos = inexistent => tc(home) = nondetect))
197                 END
198             END
199         END
200     END;
201
202     Oppositemisdeparture =
203     PRE o_mis = nondetect
204     THEN
205         o_mis := detect ||
206         IF dir = dep THEN
207             o_arr := nondetect ||
208             r_aspect := red ||
209             d_lock := lock ||
210             IF pos = arrival THEN
211                 ASSERT pos : {ct, otct, ot, between, arrival}
212                 THEN
213                     pos :(pos : POSITION &
214                     (mis = detect => (pos : {ct, otct, ot, between, arrival} & pos = ct)) &
215                     (not(mis = detect) => ((not(pos = home) => pos = inexistent) &
216                     (tc(home) = detect => pos = home) &
217                     (not(tc(home) = detect) => pos = inexistent))))
218                 END
219             END
220         END
221     END;
222
223     OppositeReleaseMisdeparture =
224     PRE o_mis = detect
225     THEN
226         o_mis := nondetect ||
```

```
227          IF r_lock = unlock & pos /: {ct, otct, ot, between, arrival} THEN
228              d_lock := unlock
229          END ||
230          IF r_lev = set & r_lock = lock & s_relay = unset &
231             (not(pos = home) => pos = inexistent) &
232             tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect & mis = nondetect
233          THEN
234              r_aspect := green
235          END
236      END;
237
238      DepartureRouteSet =
239      PRE r_lev = unset
240      THEN
241          r_lev := set ||
242          IF dir = dep & s_relay = unset THEN
243              d_lock := lock ||
244              r_lock := lock ||
245              IF tc(route) = nondetect & tc(ct) = nondetect & tc(ot) = nondetect &
246                 mis = nondetect & o_mis = nondetect &
247                  (not(pos = inexistent) => pos = home)
248              THEN r_aspect := green END
249          END
250      END;
251
252      DepartureRouteRelease =
253      PRE r_lev = set
254      THEN
255          r_lev := unset ||
256          r_aspect := red ||
257          IF r_lock = unlock & s_lev = set THEN
258             s_relay := set
259          ELSIF r_lock = lock & tc(route) = nondetect & tc(ct) = nondetect THEN
260              ASSERT not(tc(route) = detect) &
261                     not(pos = route) &
262                     d_lock = lock
263              THEN
264                  r_lock, s_relay, d_lock :(
265                      r_lock = unlock &
266                      (not(s_relay = unset) => s_relay = set) &
267                      (s_lev = unset => s_relay = unset) &
268                      (s_lev = set => s_relay = set) &
269                      d_lock : LOCK &
270                      ((r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect) =>
271                      d_lock = unlock) &
272                  (not(r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect) =>
273                      d_lock = lock))
274              END
275          END
276      END;
277
278      DepatureRouteUnlock =
279      PRE d_lock = lock & s_relay = unset & dir = dep & r_lev = unset &
280          tc(route) = nondetect & tc(ct) = nondetect
281      THEN
282          ASSERT not(pos = route) THEN
283              r_lock, s_relay, d_lock :(
284                  r_lock = unlock &
285                  (not(s_relay = unset) => s_relay = set) &
286                  (s_lev = unset => s_relay = unset) &
287                  (s_lev = set => s_relay = set) &
288                  d_lock : LOCK &
289                  ((r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect)
290                      => d_lock = unlock) &
291                  (not(r_lock = unlock & pos /: {ct, otct, ot, between, arrival} & o_mis = nondetect)
292                      => d_lock = lock)) ||
293                  r_aspect := red
294          END
295      END;
296
297      HomeTrackDetect =
298      PRE tc(home) = nondetect
299      THEN
300          ASSERT (tc <+ {home |-> detect})(route) = tc(route) &
301                 (tc <+ {home |-> detect})(home) = detect &
302                 (tc <+ {home |-> detect})(ct) = tc(ct) &
303                 (tc <+ {home |-> detect})(ot) = tc(ot) &
```

```
304                 ((r_lev = unset & (tc <+ {home |-> detect})(route) = nondetect
305                  & (tc <+ {home |-> detect})(ct) = nondetect) => r_lock = unlock)
306             THEN
307                 tc := tc <+ {home |-> detect} ||
308                 IF dir = dep & pos = inexistent THEN pos := home END
309             END
310         END;
311
312         HomeTrackNondetect =
313         PRE tc(home) = detect
314         THEN
315             ASSERT (tc <+ {home |-> nondetect})(route) = tc(route) &
316                    (tc <+ {home |-> nondetect})(home) = nondetect &
317                    (tc <+ {home |-> nondetect})(ct) = tc(ct) &
318                    (tc <+ {home |-> nondetect})(ot) = tc(ot) &
319                    ((r_lev = unset & (tc <+ {home |-> nondetect})(route) = nondetect &
320                    (tc <+ {home |-> nondetect})(ct) = nondetect) => r_lock = unlock)
321             THEN
322                 tc := tc <+ {home |-> nondetect} ||
323                 IF dir = dep & pos = home THEN pos := inexistent END
324             END
325         END;
326
327         RouteTrackDetect =
328         PRE tc(route) = nondetect
329         THEN
330             ASSERT (tc <+ {route |-> detect})(route) = detect &
331                    (tc <+ {route |-> detect})(home) = tc(home) &
332                    (tc <+ {route |-> detect})(ct) = tc(ct) &
333                    (tc <+ {route |-> detect})(ot) = tc(ot)
334             THEN
335                 tc := tc <+ {route |-> detect} ||
336                 r_aspect := red ||
337                 IF r_aspect = green THEN
338                     ASSERT not(r_lock = unlock) THEN pos := route END
339                 END
340             END
341         END;
342
343         RouteTrackNondetect =
344         PRE tc(route)= detect
345         THEN
346             ASSERT (tc <+ {route |-> nondetect})(route) = nondetect &
347                    (tc <+ {route |-> nondetect})(home) = tc(home) &
348                    (tc <+ {route |-> nondetect})(ct) = tc(ct) &
349                    (tc <+ {route |-> nondetect})(ot) = tc(ot) &
350                    not(r_aspect = green)
351             THEN
352                 tc := tc <+ {route |-> nondetect} ||
353                 IF dir = dep & tc(ct) = nondetect THEN
354                     IF r_lev = set & r_lock = lock & s_relay = unset &
355                        (not(pos = home) => pos = inexistent) &
356                        tc(ot) = nondetect & mis = nondetect & o_mis = nondetect
357                     THEN
358                         ASSERT
359                             not(r_lev = unset) &
360                             not(pos = route) &
361                             not(mis = nondetect & s_relay = set & pos = otct & tc(ot) = nondetect) &
362                             not(mis = nondetect & s_relay= set & pos = ct & tc(ot) = nondetect)
363                         THEN
364                             r_aspect := green
365                         END
366                     END||
367                     IF pos = route THEN
368                         ASSERT pos /: {ct, otct, ot, between, arrival}
369                         THEN
370                             pos:(pos : POSITION &
371                             pos /:{ct, otct, ot, between, arrival} &
372                             (not(pos = home) => pos = inexistent) &
373                             (pos = home => tc(home) = detect) &
374                             (pos = inexistent => tc(home) = nondetect))
375                         END
376                     ELSIF mis = nondetect & s_relay = set & pos = otct & tc(ot) = nondetect THEN
377                         ASSERT pos : {ct, otct, ot, between, arrival}
378                         THEN
379                             pos:(pos : POSITION &
380                             pos /:{ct, otct, ot, between, arrival} &
```

```
                        (not(pos = home) => pos = inexistent) &
                        (pos = home => tc(home) = detect) &
                        (pos = inexistent => tc(home) = nondetect))
                    END
                ELSIF mis = nondetect & s_relay= set & pos = ct & tc(ot) = nondetect THEN
                    ASSERT pos : {ct, otct, ot, between, arrival}
                    THEN
                        pos :(pos : {ct, otct, ot, between, arrival} & pos = between)
                    END
                END ||
                IF r_lev = unset THEN
                    r_lock, s_relay :(
                        r_lock = unlock &
                        (not(s_relay = unset) => s_relay = set) &
                        (s_lev = unset => s_relay = unset) &
                        (s_lev = set => s_relay = set))
                END ||
                IF r_lev = unset & o_mis = nondetect & pos /:{ct, otct, ot, between, arrival} THEN
                    d_lock := unlock
                ELSIF mis = nondetect & s_relay = set & pos = otct & tc(ot) = nondetect THEN
                    ASSERT r_lock = unlock &
                           pos : {ct, otct, ot, between, arrival}
                    THEN
                        d_lock:(d_lock : LOCK &
                            (o_mis = nondetect => d_lock = unlock) &
                            (o_mis = detect => d_lock = lock))
                    END
                END
            ELSE
                ASSERT (tc(ct) = nondetect => dir = arr) &
                       (tc(ct) = nondetect => not(r_lock = lock))
                THEN
                    skip
                END
            END
        END
    END;

    CTDetect =
    PRE tc(ct) = nondetect
    THEN
        ASSERT (tc <+ {ct |-> detect})(route) = tc(route) &
               (tc <+ {ct |-> detect})(home) = tc(home) &
               (tc <+ {ct |-> detect})(ct) = detect &
               (tc <+ {ct |-> detect})(ot) = tc(ot)
        THEN
            tc := tc <+ {ct |-> detect} ||
            r_aspect := red ||
            IF dir = dep & mis = nondetect THEN
                IF s_relay = unset THEN
                    CASE pos OF
                    EITHER home, inexistent THEN
                        ASSERT s_relay = unset => (pos /= route & pos /= ct) THEN
                            IF tc(route) = detect THEN
                                mis := detect ||
                                pos := ct ||
                                d_lock := lock
                            END
                        END
                    OR route, ct THEN
                        ASSERT not(s_relay = unset => (pos /= route & pos /= ct)) THEN
                            d_lock := lock ||
                            pos :(pos : POSITION &
                                  (not(pos = otct) => pos = ct) &
                                  (tc(ot) = detect => pos = otct) &
                                  (not(tc(ot) = detect) => pos = ct))
                        END
                    ELSE
                        ASSERT (s_relay = unset => (pos /= route & pos /= ct)) THEN
                            IF tc(route) = detect THEN mis := detect END
                        END
                    END
                    END
                ELSIF tc(route) = detect THEN
                    ASSERT (s_relay = unset => (pos /= route & pos /= ct)) THEN
                        mis := detect ||
                        IF pos : {otct, inexistent, home} THEN
```

```
                            pos := ct ||
                            d_lock := lock
                        END
                    END
                ELSIF pos = ot & tc(ot) = detect THEN pos := otct
                END
            ELSIF dir = arr & tc(route) = detect THEN mis := detect
            END
        END
    END;

    CTNondetect =
    PRE tc(ct) = detect
    THEN
        ASSERT (tc <+ {ct |-> nondetect})(route) = tc(route) &
               (tc <+ {ct |-> nondetect})(home) = tc(home) &
               (tc <+ {ct |-> nondetect})(ct) = nondetect &
               (tc <+ {ct |-> nondetect})(ot) = tc(ot) &
               not(r_aspect = green)
        THEN
            tc := tc <+ {ct |-> nondetect} ||
            IF dir = dep THEN
                IF r_lev = unset & tc(route) = nondetect THEN
                    ASSERT not(pos = route) THEN
                        r_lock, s_relay :(
                            r_lock = unlock &
                            (not(s_relay = unset) => s_relay = set) &
                            (s_lev = unset => s_relay = unset) &
                            (s_lev = set => s_relay = set))
                    END
                ELSIF r_lev = set & r_lock = lock & s_relay = unset &
                    (not(pos = home) => pos = inexistent) &
                     tc(route) = nondetect & tc(ot) = nondetect & mis = nondetect & o_mis = nondetect
                THEN
                    ASSERT not(mis = nondetect & s_relay = unset & pos = otct) &
                           not(mis = nondetect & s_relay = unset & pos = ct &
                               tc(ot) = nondetect & tc(route) = nondetect) &
                           not(mis = nondetect & s_relay = set & pos = otct &
                               tc(ot) = nondetect & tc(route) = nondetect)
                    THEN
                        r_aspect := green
                    END
                ELSE
                    ASSERT r_lev = unset => not(tc(route) = nondetect)
                    THEN
                        skip
                    END
                END ||
                IF r_lev = unset & tc(route) = nondetect & r_lock = lock & o_mis = nondetect &
                   pos /: {ct, otct, ot, between, arrival}
                THEN
                    d_lock := unlock
                ELSIF mis = nondetect & s_relay = set & pos = otct &
                      tc(ot) = nondetect & tc(route) = nondetect
                THEN
                    ASSERT r_lock = unlock &
                           pos : {ct, otct, ot, between, arrival}
                    THEN
                        d_lock:(d_lock : LOCK &
                                (o_mis = nondetect => d_lock = unlock) &
                                (o_mis = detect => d_lock = lock))
                    END
                ELSE
                    skip
                END ||
                IF mis = nondetect & s_relay = unset & pos = otct THEN
                    ASSERT pos : {ct, otct, ot, between, arrival}
                    THEN
                        pos :(pos : {ct, otct, ot, between, arrival} & pos = ot)
                    END
                ELSIF mis = nondetect & s_relay = unset & pos = ct &
                      tc(ot) = nondetect & tc(route) = nondetect
                THEN
                    ASSERT pos : {ct, otct, ot, between, arrival}
                    THEN
                        pos :(pos : {ct, otct, ot, between, arrival} & pos = between)
                    END
```

```
535                 ELSIF mis = nondetect & s_relay = set & pos = otct &
536                       tc(ot) = nondetect & tc(route) = nondetect
537                 THEN
538                     ASSERT pos : {ct, otct, ot, between, arrival}
539                     THEN
540                         pos :(pos : POSITION &
541                               pos /: {ct, otct, ot, between, arrival} &
542                               (not(pos = home) => pos = inexistent) &
543                               (pos = home => tc(home) = detect) &
544                               (pos = inexistent => tc(home) = nondetect))
545                     END
546                 END
547             ELSIF pos = ct & mis = nondetect THEN
548                 ASSERT pos : {between, approach, ot, otct, ct}
549                 THEN pos := between
550                 END
551             END
552         END
553     END;
554
555     OTDetect =
556     PRE
557         tc(ot) = nondetect
558     THEN
559         ASSERT (tc <+ {ot |-> detect})(route) = tc(route) &
560                (tc <+ {ot |-> detect})(home) = tc(home) &
561                (tc <+ {ot |-> detect})(ct) = tc(ct) &
562                (tc <+ {ot |-> detect})(ot) = detect &
563                ((r_lev = unset & (tc <+ {ot |-> detect})(route) = nondetect &
564                 (tc <+ {ot |-> detect})(ct) = nondetect) => r_lock = unlock)
565         THEN
566             tc := tc <+ {ot |-> detect} ||
567             r_aspect := red ||
568             IF dir = dep THEN
569                 IF mis = nondetect &  tc(ct) = detect THEN
570                     IF not(s_relay = unset & pos = ot) => pos : {between, arrival, ct} THEN
571                         ASSERT pos :{ct, otct, ot, between, arrival}
572                         THEN
573                             pos := otct
574                         END
575                     END
576                 END
577             ELSIF mis = nondetect & pos = approach THEN
578                 ASSERT not(o_dir = dep) THEN
579                        pos :(pos : {ct, otct, ot, between, arrival} & pos = ot)
580                 END
581             END
582         END
583     END;
584
585     OTNondetect =
586     PRE tc(ot) = detect
587     THEN
588         ASSERT (tc <+ {ot |-> nondetect})(route) = tc(route) &
589                (tc <+ {ot |-> nondetect})(home) = tc(home) &
590                (tc <+ {ot |-> nondetect})(ct) = tc(ct) &
591                (tc <+ {ot |-> nondetect})(ot) = nondetect &
592                ((r_lev = unset & (tc <+ {ot |-> nondetect})(route) = nondetect
593                & (tc <+ {ot |-> nondetect})(ct) = nondetect) =>
594                   r_lock = unlock) &
595                not(r_aspect = green)
596         THEN
597             tc := tc <+ {ot |-> nondetect} ||
598             IF dir = dep THEN
599                 IF mis = nondetect & s_relay = unset THEN
600                     IF pos = ot & tc(ct) = nondetect THEN
601                         ASSERT pos : {ct, otct, ot, between, arrival}
602                         THEN
603                             pos := between
604                         END
605                     ELSIF pos = otct THEN
606                         ASSERT pos : {ct, otct, ot, between, arrival}
607                         THEN
608                             pos := ct
609                         END
610                     ELSIF (not(pos = home) => pos = inexistent) & r_lev = set & r_lock = lock &
611                           tc(route) = nondetect & tc(ct) = nondetect & o_mis = nondetect
```

```
                    THEN r_aspect := green END
                ELSIF pos = ot THEN
                    ASSERT pos : {ct, otct, ot, between, arrival}
                    THEN
                        pos := between
                    END
                END
            ELSIF pos = otct & mis = nondetect THEN
                ASSERT pos : {ct, otct, ot, between, arrival} &    not(o_dir = dep)
                THEN
                    pos := ct
                END
            END
        END
    END;

    SubLeverSet =
    PRE s_lev = unset
    THEN
        s_lev := set ||
        IF dir = arr THEN s_relay := set
        ELSIF r_lev = unset & r_lock = unlock
        THEN
            s_relay := set ||
            ASSERT not(r_aspect = green)
            THEN
                IF dir = dep & mis = nondetect THEN
                    CASE pos OF
                    EITHER ct THEN
                        ASSERT pos : {ct, otct, ot, between, arrival}
                        THEN
                            pos :(pos : {ct, otct, ot, between, arrival} &
                            (tc(ot) = detect => pos = ot) &
                            ((not(tc(ot) = detect) & tc(ct) = nondetect & tc(route) = nondetect)
                                => pos = between) &
                            ((not(tc(ot) = detect) & not(tc(ct) = nondetect & tc(route) = nondetect))
                                => pos = ct))
                        END
                    OR ot, between, arrival THEN
                        ASSERT pos : {ct, otct, ot, between, arrival}
                        THEN
                            IF tc(ot) = detect THEN
                                pos :(pos : {ct, otct, ot, between, arrival} &
                                     (tc(ct) = detect => pos = otct) &
                                     (not(tc(ct) = detect) => pos = ot))
                            END
                        END
                    END
                    END
                END
            END
        END
    END;

    SubLeverRelease =
    PRE s_lev = set
    THEN
        s_lev := unset ||
        IF s_relay = set THEN
            s_relay := unset ||
            IF dir = dep THEN
                IF pos = otct & not(tc(ot) = detect & tc(ct) = detect) THEN
                    ASSERT pos : {ct, otct, ot, between, arrival} THEN
                           pos :( pos : {ct, otct, ot, between, arrival} & pos = ot)
                    END
                END ||
                IF r_lev = set THEN
                    r_lock := lock ||
                    d_lock := lock ||
                    IF tc(route) = nondetect & tc(ct) = nondetect &
                       tc(ot) = nondetect & o_mis = nondetect & mis = nondetect &
                       (not(pos = home) => pos = inexistent)
                    THEN r_aspect := green END
                END
            END
        END
    END;
```

```
689
690       BackwardRelease =
691       IF dir = dep & pos = ct & tc(home) = detect & tc(route) = nondetect & tc(ct) = nondetect &
692          tc(ot) = nondetect & r_lock = lock &
693         mis = nondetect
694       THEN
695           IF r_lev = set THEN
696               pos := home ||
697               IF o_mis = nondetect & mis = nondetect THEN r_aspect := green END
698           ELSE
699               r_lock, s_relay:(
700                   r_lock = unlock &
701                   s_relay : LEVER &
702                   (s_lev = unset => s_relay = unset) &
703                   (s_lev = set & r_lock = unlock & dir = arr => s_relay = set) &
704                   (s_lev = set & r_lock = unlock & r_lev = unset => s_relay = set))||
705               IF o_mis = nondetect THEN d_lock := unlock END ||
706               pos := home
707           END
708       END;
709
710       MisDepartureRelease =
711       IF mis = detect THEN
712           IF dir = dep THEN
713               IF pos = ct & tc(ct) = nondetect & tc(ot) = nondetect & tc(route) = nondetect THEN
714                   mis := nondetect ||
715                   IF r_lock = unlock & o_mis = nondetect THEN d_lock := unlock END ||
716                   pos :(pos : POSITION &
717                         pos /:{ct, otct, ot, between, arrival} &
718                         (not(pos = home) => pos = inexistent) &
719                         (pos = home => tc(home) = detect) &
720                         (pos = inexistent => tc(home) = nondetect)) ||
721                   IF r_lev = set & r_lock = lock & s_relay = unset & o_mis = nondetect
722                   THEN r_aspect := green END
723               ELSIF pos : {ot, between, arrival} &  tc(ct) = nondetect & tc(route) = nondetect THEN
724               mis := nondetect END
725           ELSIF tc(ct) = nondetect & tc(route) = nondetect THEN
726               mis := nondetect ||
727               IF pos : {ot, otct, ct} THEN
728                   ASSERT pos : {between, approach, ot, otct, ct} &
729                          pos /= between &
730                          not(o_dir = dep)
731                   THEN
732                       pos :(pos : POSITION &
733                             pos : {between, approach, ot, otct, ct} &
734                             (tc(ot) = detect => pos = ot) &
735                             (not(tc(ot) = detect) => pos = approach))
736                   END
737               END
738           END
739       END;
740
741       Approach =
742       IF dir = arr & o_dir = arr & pos = between THEN
743           ASSERT pos : {between, approach, ot, otct, ct} &
744                  not(d_lock = lock)
745           THEN pos :(pos : {between, approach, ot, otct, ct} & pos = approach)
746           END
747       END;
748
749       dd <-- GetDir =
750       BEGIN dd := dir END;
751
752       pp <-- GetPos =
753       BEGIN pp := pos END;
754
755       mm <-- GetMis =
756       BEGIN mm := mis END;
757
758       SetParam(r_le, r_lo, r_as, s_le, s_re, d_le, di, d_lo, o_di, o_ar,
759                    o_mi, po, th, tr, tct, to, mi, ap) =
760       PRE
761           r_le : LEVER & r_lo : LOCK & r_as : ASPECT & s_le : LEVER & s_re : LEVER &
762           d_le : DIRECTION & di : DIRECTION & d_lo : LOCK & o_di : DIRECTION & o_ar : DETECT &
763           o_mi : DETECT & po : POSITION & th : DETECT & tr : DETECT & tct : DETECT & to : DETECT &
764           mi : DETECT & ap : DETECT &
765           (di = dep =>
```

```
766             (po : {inexistent, home, route, ct, otct, ot, between, arrival} &
767             (po = arrival => (o_ar = detect & o_mi = nondetect)) &
768             (po : {ct, otct, ot, between, arrival} => d_lo = lock) &
769             (po = home => th = detect) &
770             (po = inexistent => th = nondetect) &
771             (po = otct => (s_re = unset => (to = detect & tct = detect))) &
772             (po = otct => mi = nondetect) &
773             (po = route => (r_lo = lock & tr = detect & tct = nondetect)) &
774             (po /: {ct, otct, ot, between, arrival} => mi = nondetect))) &
775
776             (r_lo = lock => di = dep) &
777             (r_lo = lock => d_lo = lock) &
778             (r_lo = lock => s_re = unset) &
779             ((r_le = unset & tr = nondetect & tct = nondetect) => r_lo = unlock) &
780             ((di = dep & r_le = set & s_re = unset) => r_lo = lock) &
781
782             (d_lo = lock => di = dep) &
783             (di = dep & o_mi = detect => d_lo = lock) &
784             ((r_lo = unlock & po /: {ct, otct, ot, between, arrival} & o_mi = nondetect) =>
785                 d_lo = unlock) &
786         (di = dep => o_di = dep) &
787         (di = arr =>
788             (po : {between, approach, ot, otct, ct} &
789             (o_di = dep => po = between))) &
790
791         (r_as = green =>
792             (r_le = set & r_lo = lock & s_re = unset &
793             (not(po = home) => po = inexistent) &
794              tr = nondetect & tct = nondetect & to = nondetect & mi = nondetect & o_mi = nondetect)) &
795
796         (s_le = unset => s_re = unset) &
797         (s_le = set & r_lo = unlock & di = arr => s_re = set) &
798         (s_le = set & r_lo = unlock & r_le = unset => s_re = set)
799     THEN
800         r_lev := r_le ||
801         r_lock := r_lo ||
802         r_aspect := r_as ||
803         s_lev := s_le ||
804         s_relay := s_re ||
805         d_lev := d_le ||
806         dir := di ||
807         d_lock := d_lo ||
808         o_dir := o_di ||
809         o_arr := o_ar ||
810         o_mis := o_mi ||
811         pos := po ||
812         tc := {home |-> th, route |-> tr, ct |-> tct, ot |-> to} ||
813         mis := mi ||
814         app := ap
815     END
816
817 END
```

付録 D

ブレーキ計算モデル（B）

D.1 共通定義

D.1.1 types.def

```
DEFINITIONS
    SPEED == (0..MaxSpeed);
    GRAD == (0..MaxGrad);
    LIMIT == SPEED * NAT * GRAD;
    DECL == (1..255);

    Gr1 == 353;
    Gr2 == 1000
```

D.1.2 Parameter1.mch

```
MACHINE
    Parameter1

DEFINITIONS
    "types.def"

CONSTANTS
    MaxSpeed, MaxGrad, MaxDist, n_decl, e_decl

PROPERTIES
    MaxSpeed : NAT1 &
    MaxSpeed * MaxSpeed < MAXINT / 16 &
    MaxGrad : 1..255 &
    MaxDist : NAT1 &
    MaxDist < MAXINT / 4 &
    e_decl: 1..255 &
    n_decl: 1..255 &
    n_decl <= e_decl &
    n_decl * 72 > MaxGrad * Gr1 *72 / Gr2

END
```

D.1.3 Parameter2.mch

```
MACHINE
    Parameter2

SEES
    Parameter1

CONSTANTS
    warning_release

PROPERTIES
    warning_release : 0..MaxSpeed
```

```
12
13  END
```

D.1.4　Parameter3.mch

```
1   MACHINE
2       Parameter3
3
4   CONSTANTS
5       cinterval, cdelay_e, cdelay_n, cdelay_w, c_margin_e, c_margin_n, c_margin_w
6
7   PROPERTIES
8       cinterval : 0..16383 &
9       cdelay_e : 0..16383 &
10      cdelay_n : 0..16383 &
11      cdelay_w : 0..16383 &
12      cinterval + cdelay_e : 0..16383 &
13      cinterval + cdelay_n : 0..16383 &
14      cinterval + cdelay_w : 0..16383 &
15      cdelay_e <= cdelay_n &
16      cdelay_n <= cdelay_w &
17
18      c_margin_e : 0..8191 &
19      c_margin_n : 0..8191 &
20      c_margin_w : 0..8191 &
21
22      c_margin_n >= c_margin_e &
23      c_margin_w >= c_margin_n
24
25  ASSERTIONS
26      !lm.(lm : 0..32767 => lm * (cinterval + cdelay_e) / 36000 <= 32767);
27      !lm.(lm : 0..32767 => lm * (cinterval + cdelay_n) / 36000 <= 32767);
28      !lm.(lm : 0..32767 => lm * (cinterval + cdelay_w) / 36000 <= 32767);
29
30      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
31          (lm1 <= lm2 => lm1 * (cinterval + cdelay_e) / 36000 <= lm2 * (cinterval + cdelay_e) / 36000));
32      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
33          (lm1 <= lm2 => lm1 * (cinterval + cdelay_n) / 36000 <= lm2 * (cinterval + cdelay_n) / 36000));
34      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
35          (lm1 <= lm2 => lm1 * (cinterval + cdelay_w) / 36000 <= lm2 * (cinterval + cdelay_w) / 36000));
36
37      !lm.(lm : 0..32767 => lm * (cinterval + cdelay_e) / 36000 <= lm * (cinterval + cdelay_n) / 36000);
38      !lm.(lm : 0..32767 => lm * (cinterval + cdelay_n) / 36000 <= lm * (cinterval + cdelay_w) / 36000)
39
40  END
```

D.2　BrakeSpeed マシン

D.2.1　BrakeSpeed.mch（抽象機械）

```
1   MACHINE
2       BrakeSpeed
3
4   SEES
5       Parameter1
6
7   DEFINITIONS
8       "types.def"
9
10  ABSTRACT_CONSTANTS
11      release
12
13  PROPERTIES
14      release : SPEED
15
16  VARIABLES
17      emergencyspeed, normalspeed, warningspeed, releasespeed,
18      target, location, grad, e_distance, n_distance, w_distance
19
20  INVARIANT
21      target : NAT &
```

```
22      location : NAT &
23      grad : GRAD &
24
25      emergencyspeed : SPEED &
26      normalspeed : SPEED &
27      warningspeed : SPEED &
28      releasespeed : SPEED &
29      e_distance : 0..(MaxSpeed * MaxSpeed) &
30      n_distance : 0..(MaxSpeed * MaxSpeed) &
31      w_distance : 0..(MaxSpeed * MaxSpeed) &
32
33      (target <= location =>
34          (emergencyspeed = 0 &
35           normalspeed = 0 &
36           warningspeed = 0 &
37           releasespeed = 0)) &
38
39      (not(target <= location) =>
40              (e_distance <= target - location) &
41              (n_distance <= target - location) &
42              (w_distance <= target - location)) &
43
44      (emergencyspeed = 0 => e_distance = 0) &
45      (normalspeed = 0 => n_distance = 0) &
46      (warningspeed = 0 => w_distance = 0) &
47
48      normalspeed <= emergencyspeed &
49      warningspeed <= normalspeed &
50      releasespeed = max({warningspeed - release, 0})
51
52  INITIALISATION
53      grad := 0 ||
54      ANY tt WHERE tt : NAT
55      THEN
56          target := tt ||
57          location := tt
58      END ||
59      normalspeed := 0 ||
60      emergencyspeed := 0 ||
61      warningspeed := 0 ||
62      releasespeed := 0 ||
63      e_distance := 0 ||
64      n_distance := 0 ||
65      w_distance := 0
66
67  OPERATIONS
68      SetSpeed(ta, lo, gr) =
69      PRE
70          ta : NAT &
71          gr : GRAD &
72          lo : NAT
73      THEN
74          target := ta ||
75          grad := gr ||
76          location := lo ||
77          IF ta <= lo THEN
78              emergencyspeed := 0 ||
79              normalspeed := 0 ||
80              warningspeed := 0 ||
81              releasespeed := 0 ||
82              n_distance := 0 ||
83              e_distance := 0 ||
84              w_distance := 0
85          ELSE
86              emergencyspeed, normalspeed, releasespeed, warningspeed,
87              e_distance, n_distance, w_distance :(
88                  e_distance : 0..(MaxSpeed * MaxSpeed) &
89                  n_distance : 0..(MaxSpeed * MaxSpeed) &
90                  w_distance : 0..(MaxSpeed * MaxSpeed) &
91                  emergencyspeed : SPEED &
92                  normalspeed : SPEED &
93                  releasespeed : SPEED &
94                  warningspeed : SPEED &
95
96                  e_distance <= ta - lo &
97                  n_distance <= ta - lo &
98                  w_distance <= ta - lo &
```

```
                (emergencyspeed = 0 => e_distance = 0) &
                (normalspeed = 0 => n_distance = 0) &
                (warningspeed = 0 => w_distance = 0) &
                normalspeed <= emergencyspeed &
                warningspeed <= normalspeed &
                releasespeed = max({warningspeed - releaSe, 0})
            )
        End
    End;

    Sp <-- Normalspeed =
        Sp := Normalspeed;

    Sp <-- Emergencyspeed =
        Sp := Emergencyspeed;

    Sp <-- Warningspeed =
        Sp := Warningspeed;

    Sp <-- Releasespeed =
        sp := releasespeed

END
```

D.2.2　BrakeSpeed_1.ref（詳細化 1）

```
REFINEMENT
    BrakeSpeed_1

REFINES
    BrakeSpeed

SEES
    Parameter1

DEFINITIONS
    "types.def"

ABSTRACT_CONSTANTS
    margin, release

PROPERTIES
    margin : NAT &
    margin : 0..16383

VARIABLES
    emergencyspeed, normalspeed, warningspeed,
    target, location, grad, e_distance, n_distance, w_distance

INVARIANT
    (target <= location + margin =>
        (emergencyspeed = 0 &
         normalspeed = 0 &
         warningspeed = 0)) &

    (not(target <= location + margin) =>
        (e_distance <= target - location - margin &
         n_distance <= target - location - margin &
         w_distance <= target - location - margin))

ASSERTIONS
    max({warningspeed - release, 0}) : 0..MaxSpeed

INITIALISATION
    grad := 0 ||
    ANY tt WHERE tt : NAT
    THEN
        target := tt ||
        location := tt
    END ||
    normalspeed := 0 ||
    emergencyspeed := 0 ||
    warningspeed := 0 ||
    e_distance := 0 ||
    n_distance := 0 ||
```

```
50      w_distance := 0
51
52  OPERATIONS
53      SetSpeed(ta, lo, gr) =
54      PRE
55          ta : NAT &
56          gr : GRAD &
57          lo : NAT
58      THEN
59          target := ta ||
60          grad := gr ||
61          location := lo ||
62          IF ta <= lo + margin THEN
63              emergencyspeed := 0 ||
64              normalspeed := 0 ||
65              warningspeed := 0 ||
66              n_distance := 0 ||
67              e_distance := 0 ||
68              w_distance := 0
69          ELSE
70              ASSERT not(ta <= lo)
71              THEN
72                  emergencyspeed, normalspeed, warningspeed, e_distance, n_distance, w_distance :(
73                      emergencyspeed : SPEED &
74                      normalspeed : SPEED &
75                      warningspeed : SPEED &
76                      e_distance : 0..(MaxSpeed * MaxSpeed) &
77                      n_distance : 0..(MaxSpeed * MaxSpeed) &
78                      w_distance : 0..(MaxSpeed * MaxSpeed) &
79                      e_distance <= ta - lo - margin &
80                      n_distance <= ta - lo - margin &
81                      w_distance <= ta - lo - margin &
82                      (emergencyspeed = 0 => e_distance = 0) &
83                      (normalspeed = 0 => n_distance = 0) &
84                      (warningspeed = 0 => w_distance = 0) &
85                      normalspeed <= emergencyspeed &
86                      warningspeed <= normalspeed
87                  )
88              END
89          END
90      END;
91
92      sp <-- Normalspeed =
93          sp := normalspeed;
94
95      sp <-- Emergencyspeed =
96          sp := emergencyspeed;
97
98      sp <-- Warningspeed =
99          sp := warningspeed;
100
101     sp <-- Releasespeed =
102         sp := max({warningspeed - release, 0})
103
104 END
```

D.2.3 BrakeSpeed_2.ref（詳細化 2）

```
1   REFINEMENT
2       BrakeSpeed_2
3
4   REFINES
5       BrakeSpeed_1
6
7   SEES
8       Parameter1
9
10  DEFINITIONS
11      "types.def"
12
13  ABSTRACT_CONSTANTS
14       margin, delay, release
15
16  PROPERTIES
17      delay : NAT &
```

```
    delay : 0..16383

VARIABLES
    emergencyspeed, normalspeed, warningspeed,
    target, location, grad, e_distance, n_distance, w_distance

INVARIANT
    (not(target <= location + margin) =>
        (emergencyspeed * delay / 36000 + e_distance <= target - location - margin &
         normalspeed * delay / 36000 + n_distance <= target - location - margin &
         warningspeed * delay / 36000 + w_distance <= target - location - margin))

INITIALISATION
    grad := 0 ||
    ANY tt WHERE tt : NAT
    THEN
        target := tt ||
        location := tt
    END ||
    normalspeed := 0 ||
    emergencyspeed := 0 ||
    warningspeed := 0 ||
    e_distance := 0 ||
    n_distance := 0 ||
    w_distance := 0

OPERATIONS
    SetSpeed(ta, lo, gr) =
    PRE
        ta : NAT &
        gr : GRAD &
        lo : NAT
    THEN
        target := ta ||
        grad := gr ||
        location := lo ||
        IF ta <= lo + margin THEN
            emergencyspeed := 0 ||
            normalspeed := 0 ||
            warningspeed := 0 ||
            e_distance := 0 ||
            n_distance := 0 ||
            w_distance := 0
        ELSE
            emergencyspeed, normalspeed, warningspeed,
            e_distance, n_distance, w_distance :(
                emergencyspeed : SPEED &
                normalspeed : SPEED &
                warningspeed : SPEED &
                e_distance : 0..(MaxSpeed * MaxSpeed) &
                n_distance : 0..(MaxSpeed * MaxSpeed) &
                w_distance : 0..(MaxSpeed * MaxSpeed) &
                emergencyspeed * delay / 36000 + e_distance <= ta - lo - margin &
                normalspeed * delay / 36000 + n_distance <= ta - lo - margin &
                warningspeed * delay / 36000 + w_distance <= ta - lo - margin &
                (emergencyspeed = 0 => e_distance = 0) &
                (normalspeed = 0 => n_distance = 0) &
                (warningspeed = 0 => w_distance = 0) &
                normalspeed <= emergencyspeed &
                warningspeed <= normalspeed
            )
        END
    END;

    sp <-- Normalspeed =
        sp := normalspeed;

    sp <-- Emergencyspeed =
        sp := emergencyspeed;

    sp <-- Warningspeed =
        sp := warningspeed;

    sp <-- Releasespeed =
        sp := max({warningspeed - release, 0})

END
```

D.2.4 BrakeSpeed_3.ref（詳細化 3）

```
REFINEMENT
    BrakeSpeed_3

REFINES
    BrakeSpeed_2

SEES
    Parameter1

DEFINITIONS
    "types.def"

ABSTRACT_CONSTANTS
    interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release

PROPERTIES
    interval : NAT &
    interval : 0..16383 &
    bdelay_e : 0..16383 &
    bdelay_n : 0..16383 &
    bdelay_w : 0..16383 &
    interval + bdelay_e = delay  &/* delay in msec */
    bdelay_e <= bdelay_n &
    bdelay_n <= bdelay_w &

    margin_e : NAT & margin_e : 0..16383 &
    margin_n : NAT & margin_n : 0..16383 &
    margin_w : NAT & margin_w : 0..16383 &

    margin_e = margin &
    margin_n >= margin_e &
    margin_w >= margin_n

ASSERTIONS
    !sp.(sp : SPEED =>
        sp * (interval + bdelay_e)/36000 <= sp * (interval + bdelay_n)/36000);
    !sp.(sp : SPEED =>
        sp * (interval + bdelay_n)/36000 <= sp * (interval + bdelay_w)/36000)

VARIABLES
    emergencyspeed, normalspeed, warningspeed,
    target, location, grad, e_distance, n_distance, w_distance

INVARIANT
    (target <= location + margin_e => emergencyspeed = 0) &
    (target <= location + margin_n => normalspeed = 0) &
    (target <= location + margin_w => warningspeed = 0) &

    (not(target <= location + margin_e) =>
        emergencyspeed * (interval + bdelay_e) / 36000 + e_distance <= target - location - margin_e) &
    (not(target <= location + margin_n) =>
         normalspeed * (interval + bdelay_n) / 36000 + n_distance <= target - location - margin_n) &
    (not(target <= location + margin_w) =>
         warningspeed * (interval + bdelay_w) / 36000 +    w_distance <= target - location - margin_w)

INITIALISATION
    grad := 0 ||
    ANY tt WHERE tt : NAT
    THEN
        target := tt ||
        location := tt
    END ||
    normalspeed := 0 ||
    emergencyspeed := 0 ||
    warningspeed := 0 ||
    e_distance := 0 ||
    n_distance := 0 ||
    w_distance := 0

OPERATIONS
    SetSpeed(ta, lo, gr) =
    PRE
```

```
73          ta : NAT &
74          gr : GRAD &
75          lo : NAT
76      THEN
77          target := ta ||
78          grad := gr ||
79          location := lo ||
80          IF ta <= lo + margin_e THEN
81              emergencyspeed := 0 ||
82              normalspeed := 0 ||
83              warningspeed := 0 ||
84              e_distance := 0 ||
85              n_distance := 0 ||
86              w_distance := 0
87          ELSE
88              emergencyspeed, normalspeed, warningspeed, e_distance, n_distance, w_distance :(
89                  emergencyspeed : SPEED &
90                  normalspeed : SPEED &
91                  warningspeed : SPEED &
92                  e_distance : 0..(MaxSpeed * MaxSpeed) &
93                  n_distance : 0..(MaxSpeed * MaxSpeed) &
94                  w_distance : 0..(MaxSpeed * MaxSpeed) &
95                  emergencyspeed : SPEED &
96                  normalspeed : SPEED &
97                  warningspeed : SPEED &
98
99                  (ta <= lo + margin_e => emergencyspeed = 0) &
100                 (not(ta <= lo + margin_e) =>
101                 emergencyspeed * (interval + bdelay_e) / 36000 + e_distance <= ta - lo - margin_e) &
102                 (ta <= lo + margin_n => normalspeed = 0) &
103                 (not(ta <= lo + margin_n) =>
104                 normalspeed * (interval + bdelay_n) / 36000 + n_distance <= ta - lo - margin_n) &
105                 (ta <= lo + margin_w => warningspeed = 0) &
106                 (not(ta <= lo + margin_w) =>
107                 warningspeed * (interval + bdelay_w) / 36000 + w_distance <= ta - lo - margin_w) &
108
109                 (emergencyspeed = 0 => e_distance = 0) &
110                 (normalspeed = 0 => n_distance = 0) &
111                 (warningspeed = 0 => w_distance = 0) &
112
113                 normalspeed <= emergencyspeed &
114                 warningspeed <= normalspeed
115             )
116         END
117     END;
118
119     sp <-- Normalspeed =
120         sp := normalspeed;
121
122     sp <-- Emergencyspeed =
123         sp := emergencyspeed;
124
125     sp <-- Warningspeed =
126         sp := warningspeed;
127
128     sp <-- Releasespeed =
129         sp := max({warningspeed - release, 0})
130
131 END
```

D.2.5　BrakeSpeed_4.ref（詳細化 4）

```
1   REFINEMENT
2       BrakeSpeed_4
3
4   REFINES
5       BrakeSpeed_3
6
7   SEES
8       Parameter1
9
10  DEFINITIONS
11      "types.def"
12
13  ABSTRACT_CONSTANTS
```

```
14      interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release
15
16  VARIABLES
17      emergencyspeed, normalspeed, warningspeed, loc, grad, e_distance, n_distance, w_distance
18
19  INVARIANT
20      loc : NAT &
21      (target < location => loc = 0) &
22      (not(target < location) => loc = target - location) &
23
24      (loc <= margin_e => emergencyspeed = 0) &
25      (loc <= margin_n => normalspeed = 0) &
26      (loc <= margin_w => warningspeed = 0) &
27
28      (not(loc <= margin_e) =>
29          emergencyspeed * (interval + bdelay_e) / 36000 + e_distance <= loc - margin_e) &
30
31      (not(loc <= margin_n) =>
32           normalspeed * (interval + bdelay_n) / 36000 + n_distance <= loc - margin_n) &
33
34      (not(loc <= margin_w) =>
35           warningspeed * (interval + bdelay_w) / 36000 +    w_distance <= loc - margin_w)
36
37  INITIALISATION
38      grad := 0 ||
39      loc := 0 ||
40      normalspeed := 0 ||
41      emergencyspeed := 0 ||
42      warningspeed := 0 ||
43      e_distance := 0 ||
44      n_distance := 0 ||
45      w_distance := 0
46
47  OPERATIONS
48      SetSpeed(ta, lo, gr) =
49      PRE
50          ta : NAT &
51          gr : GRAD &
52          lo : NAT
53      THEN
54          grad := gr ||
55          IF ta < lo THEN
56              loc := 0 ||
57              emergencyspeed := 0 ||
58              normalspeed := 0 ||
59              warningspeed := 0  ||
60              e_distance := 0 ||
61              n_distance := 0 ||
62              w_distance := 0
63          ELSE
64              ANY ll WHERE ll : NAT & ll = ta - lo
65              THEN
66                  loc := ll ||
67                  IF ll <= margin_e THEN
68                      emergencyspeed := 0 ||
69                      normalspeed := 0 ||
70                      warningspeed := 0 ||
71                      e_distance := 0 ||
72                      n_distance := 0 ||
73                      w_distance := 0
74                  ELSE
75                      emergencyspeed, normalspeed, warningspeed, e_distance, n_distance, w_distance :(
76                          emergencyspeed : SPEED &
77                          normalspeed : SPEED &
78                          warningspeed : SPEED &
79                          e_distance : 0..(MaxSpeed * MaxSpeed) &
80                          n_distance : 0..(MaxSpeed * MaxSpeed) &
81                          w_distance : 0..(MaxSpeed * MaxSpeed) &
82
83                          (ll <= margin_e => emergencyspeed = 0) &
84                          (not(ll <= margin_e) =>
85                              emergencyspeed * (interval + bdelay_e) / 36000 +  e_distance <= ll - margin_e) &
86                          (ll <= margin_n => normalspeed = 0) &
87                          (not(ll <= margin_n) =>
88                              normalspeed * (interval + bdelay_n) / 36000 +  n_distance <= ll - margin_n) &
89                          (ll <= margin_w => warningspeed = 0) &
90                          (not(ll <= margin_w) =>
```

```
                                warningspeed * (interval + bdelay_w) / 36000 + w_distance <= ll - margin_w) &

                            (emergencyspeed = 0 => e_distance = 0) &
                            (normalspeed = 0 => n_distance = 0) &
                            (warningspeed = 0 => w_distance = 0) &

                            normalspeed <= emergencyspeed &
                            warningspeed <= normalspeed
                        )
                        END
                    END
                END
            END;

        sp <-- Normalspeed =
            sp := normalspeed;

        sp <-- Emergencyspeed =
            sp := emergencyspeed;

        sp <-- Warningspeed =
            sp := warningspeed;

        sp <-- Releasespeed =
            sp := max({warningspeed - release, 0})

    END
```

D.2.6　BrakeSpeed_5.ref（詳細化 5）

```
REFINEMENT
    BrakeSpeed_5

REFINES
    BrakeSpeed_4

SEES
    Parameter1

DEFINITIONS
    "types.def"

ABSTRACT_CONSTANTS
    interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release

VARIABLES
    emergencyspeed, normalspeed, warningspeed, loc2, grad, e_distance, n_distance, w_distance

INVARIANT
    loc2 : 0..MaxDist &
    (loc <= MaxDist => loc2 = loc) &
    (not(loc <= MaxDist) => loc2 = MaxDist) &

    (loc2 <= margin_e => emergencyspeed = 0) &
    (loc2 <= margin_n => normalspeed = 0) &
    (loc2 <= margin_w => warningspeed = 0) &

    (not(loc2 <= margin_e) =>
        emergencyspeed * (interval + bdelay_e) / 36000 + e_distance <= loc2 - margin_e) &

    (not(loc2 <= margin_n) =>
        normalspeed * (interval + bdelay_n) / 36000 + n_distance <= loc2 - margin_n) &

    (not(loc2 <= margin_w) =>
        warningspeed * (interval + bdelay_w) / 36000 + w_distance <= loc2 - margin_w)

INITIALISATION
    grad := 0 ||
    loc2 := 0 ||
    normalspeed := 0 ||
    emergencyspeed := 0 ||
    warningspeed := 0 ||
    e_distance := 0 ||
    n_distance := 0 ||
    w_distance := 0
```

```
46
47  OPERATIONS
48      SetSpeed(ta, lo, gr) =
49      PRE
50          ta : NAT &
51          gr : GRAD &
52          lo : NAT
53      THEN
54          grad := gr ||
55          IF ta < lo THEN
56              loc2 := 0 ||
57              emergencyspeed := 0 ||
58              normalspeed := 0 ||
59              warningspeed := 0 ||
60              e_distance := 0 ||
61              n_distance := 0 ||
62              w_distance := 0
63          ELSE
64              ANY ll WHERE
65                  ll : NAT &
66                  (ta - lo <= MaxDist => ll = ta - lo) &
67                  (not(ta - lo <= MaxDist) => ll = MaxDist)
68              THEN
69                  loc2 := ll ||
70                  IF ll <= margin_e THEN
71                      emergencyspeed := 0 ||
72                      normalspeed := 0 ||
73                      warningspeed := 0 ||
74                      e_distance := 0 ||
75                      n_distance := 0 ||
76                      w_distance := 0
77                  ELSE
78                      emergencyspeed, normalspeed, warningspeed, e_distance, n_distance, w_distance :(
79                          emergencyspeed : SPEED &
80                          normalspeed : SPEED &
81                          warningspeed : SPEED &
82                          e_distance : 0..(MaxSpeed * MaxSpeed) &
83                          n_distance : 0..(MaxSpeed * MaxSpeed) &
84                          w_distance : 0..(MaxSpeed * MaxSpeed) &
85
86                          (ll <= margin_e => emergencyspeed = 0) &
87                          (not(ll <= margin_e) =>
88                              emergencyspeed * (interval + bdelay_e) / 36000 + e_distance <= ll - margin_e) &
89
90                          (ll <= margin_n => normalspeed = 0) &
91                          (not(ll <= margin_n) =>
92                              normalspeed * (interval + bdelay_n) / 36000 + n_distance <= ll - margin_n) &
93
94                          (ll <= margin_w => warningspeed = 0) &
95                          (not(ll <= margin_w) =>
96                              warningspeed * (interval + bdelay_w) / 36000 + w_distance <= ll - margin_w) &
97
98                          (emergencyspeed = 0 => e_distance = 0) &
99                          (normalspeed = 0 => n_distance = 0) &
100                         (warningspeed = 0 => w_distance = 0) &
101
102                         normalspeed <= emergencyspeed &
103                         warningspeed <= normalspeed
104                     )
105                 END
106             END
107         END
108     END;
109
110     sp <-- Normalspeed =
111         sp := normalspeed;
112
113     sp <-- Emergencyspeed =
114         sp := emergencyspeed;
115
116     sp <-- Warningspeed =
117         sp := warningspeed;
118
119     sp <-- Releasespeed =
120         sp := max({warningspeed - release, 0})
121
122 END
```

D.2.7　BrakeSpeed_6.ref（詳細化 6）

```
REFINEMENT
    BrakeSpeed_6

REFINES
    BrakeSpeed_5

SEES
    Parameter1

DEFINITIONS
    "types.def"

INCLUDES
    Distance

ABSTRACT_CONSTANTS
    interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release

VARIABLES
    emergencyspeed, normalspeed, warningspeed, loc2, grad

INVARIANT
    e_distance = distance_e(emergencyspeed, grad) &
    n_distance = distance_n(normalspeed, grad) &
    w_distance = distance_n(warningspeed, grad)

INITIALISATION
    grad := 0 ||
    loc2 := 0 ||
    normalspeed := 0 ||
    emergencyspeed := 0 ||
    warningspeed := 0

OPERATIONS
    SetSpeed(ta, lo, gr) =
    PRE
        ta : NAT &
        gr : GRAD &
        lo : NAT
    THEN
        grad := gr ||
        IF ta < lo THEN
            loc2 := 0 ||
            emergencyspeed := 0 ||
            normalspeed := 0 ||
            warningspeed := 0
        ELSE
            ANY ll WHERE
                ll : NAT &
                (ta - lo <= MaxDist => ll = ta - lo) &
                (not(ta - lo <= MaxDist) => ll = MaxDist)
            THEN
                loc2 := ll ||
                IF ll <= margin_e THEN
                    emergencyspeed := 0 ||
                    normalspeed := 0 ||
                    warningspeed := 0
                ELSE
                    emergencyspeed, normalspeed, warningspeed :(
                        emergencyspeed : SPEED &
                        normalspeed : SPEED &
                        warningspeed : SPEED &

                        (ll <= margin_e => emergencyspeed = 0) &
                        (not(ll <= margin_e) =>
                            emergencyspeed * (interval + bdelay_e) / 36000 + distance_e(emergencyspeed, gr)
                                <= ll - margin_e) &

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

```
73
74                          (ll <= margin_w => warningspeed = 0) &
75                          (not(ll <= margin_w) =>
76                              warningspeed * (interval + bdelay_w) / 36000 + distance_n(warningspeed, gr)
77                                  <= ll - margin_w) &
78
79                          normalspeed <= emergencyspeed &
80                          warningspeed <= normalspeed
81                      )
82                  END
83              END
84          END
85      END;
86
87      sp <-- Normalspeed =
88          sp := normalspeed;
89
90      sp <-- Emergencyspeed =
91          sp := emergencyspeed;
92
93      sp <-- Warningspeed =
94          sp := warningspeed;
95
96      sp <-- Releasespeed =
97          IF warningspeed - release >= 0 THEN
98              sp := warningspeed - release
99          ELSE
100             sp := 0
101         END
102
103 END
```

D.2.8 BrakeSpeed_7.ref（詳細化 7）

```
1   REFINEMENT
2       BrakeSpeed_7
3
4   REFINES
5       BrakeSpeed_6
6
7   SEES
8       Parameter1
9
10  DEFINITIONS
11      "types.def"
12
13  INCLUDES
14      Distance
15
16  ABSTRACT_CONSTANTS
17      interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release
18
19  VARIABLES
20      emergencyspeed, normalspeed, warningspeed, loc2, grad
21
22  INVARIANT
23      (not(loc2 <= margin_e) =>
24          emergencyspeed = max({sp | sp : SPEED &
25              sp * (interval + bdelay_e) / 36000 + distance_e(sp,grad) <= loc2 - margin_e})) &
26      (not(loc2 <= margin_n) =>
27           normalspeed = max({sp | sp : SPEED &
28              sp * (interval + bdelay_n) / 36000 + distance_n(sp,grad) <= loc2 - margin_n})) &
29      (not(loc2 <= margin_w) =>
30           warningspeed = max({sp | sp : SPEED &
31              sp * (interval + bdelay_w) / 36000 + distance_n(sp,grad) <= loc2 - margin_w}))
32
33  INITIALISATION
34      grad := 0 ||
35      loc2 := 0 ||
36      normalspeed := 0 ||
37      emergencyspeed := 0 ||
38      warningspeed := 0
39
40  OPERATIONS
41      SetSpeed(ta, lo, gr) =
```

```
42      PRE
43          ta : NAT &
44          gr : GRAD &
45          lo : NAT
46      THEN
47          grad := gr ||
48          IF ta < lo THEN
49              loc2 := 0 ||
50              emergencyspeed := 0 ||
51              normalspeed := 0 ||
52              warningspeed := 0
53          ELSE
54              ANY ll WHERE
55                  ll : NAT &
56                  (ta - lo <= MaxDist => ll = ta - lo) &
57                  (not(ta - lo <= MaxDist) => ll = MaxDist)
58              THEN
59                  loc2 := ll ||
60                  IF ll <= margin_e THEN
61                      emergencyspeed := 0
62                  ELSE
63                      emergencyspeed := max({sp | sp : SPEED &
64                          sp * (interval + bdelay_e) / 36000 + distance_e(sp, gr) <= ll - margin_e})
65                  END ||
66                  IF ll <= margin_n THEN
67                      normalspeed := 0
68                  ELSE
69                      normalspeed := max({sp | sp : SPEED &
70                          sp * (interval + bdelay_n) / 36000 + distance_n(sp, gr) <= ll - margin_n})
71                  END ||
72                  IF ll <= margin_w THEN
73                      warningspeed := 0
74                  ELSE
75                      warningspeed := max({sp | sp : SPEED &
76                          sp * (interval + bdelay_w) / 36000 + distance_n(sp, gr) <= ll - margin_w})
77                  END
78              END
79          END
80      END;
81
82      sp <-- Normalspeed =
83          sp := normalspeed;
84
85      sp <-- Emergencyspeed =
86          sp := emergencyspeed;
87
88      sp <-- Warningspeed =
89          sp := warningspeed;
90
91      sp <-- Releasespeed =
92          IF warningspeed - release >= 0
93          THEN
94              sp := warningspeed - release
95          ELSE
96              sp := 0
97          END
98
99  END
```

D.2.9　BrakeSpeed_8.ref（詳細化 8）

```
1   REFINEMENT
2       BrakeSpeed_8
3
4   REFINES
5       BrakeSpeed_7
6
7   INCLUDES
8       Distance
9
10  SEES
11      Parameter1
12
13  DEFINITIONS
14      "types.def"
```

```
15
16  ABSTRACT_CONSTANTS
17      interval, bdelay_e, bdelay_n, bdelay_w, margin_n, margin_e, margin_w, release
18
19  VARIABLES
20      loc2, grad
21
22  INITIALISATION
23      grad := 0 ||
24      loc2 := 0
25
26  OPERATIONS
27      SetSpeed(ta, lo, gr) =
28      PRE
29          ta : NAT &
30          gr : GRAD &
31          lo : NAT
32      THEN
33          grad := gr ||
34          IF ta < lo
35          THEN
36              loc2 := 0
37          ELSIF ta - lo <= MaxDist
38          THEN
39              loc2 :=  ta - lo
40          ELSE
41              loc2 := MaxDist
42          END
43      END;
44
45      sp <-- Emergencyspeed =
46      IF loc2 <= margin_e THEN
47          sp := 0
48      ELSE
49          sp := max({sp1 | sp1 : SPEED &
50              sp1 * (interval + bdelay_e) / 36000 + distance_e(sp1, grad) <= loc2 - margin_e})
51      END;
52
53      sp <-- Normalspeed =
54      IF loc2 <= margin_n THEN
55          sp := 0
56      ELSE
57          sp := max({sp1 | sp1 : SPEED &
58              sp1 * (interval + bdelay_n) / 36000 + distance_n(sp1, grad) <= loc2 - margin_n})
59      END;
60
61      sp <-- Warningspeed =
62      IF loc2 <= margin_w THEN
63          sp := 0
64      ELSE
65          sp := max({sp1 | sp1 : SPEED &
66              sp1 * (interval + bdelay_w) / 36000 + distance_n(sp1, grad) <= loc2 - margin_w})
67      END;
68
69      sp <-- Releasespeed =
70      IF loc2 <= margin_w THEN
71          sp := 0
72      ELSE
73          ANY xx WHERE
74              xx : NAT &
75              xx = max({sp1 | sp1 : SPEED &
76                  sp1 * (interval + bdelay_w) / 36000 + distance_n(sp1, grad) <= loc2 - margin_w})
77          THEN
78              IF xx - release >= 0 THEN
79                  sp := xx - release
80              ELSE
81                  sp := 0
82              END
83          END
84      END
85
86  END
```

D.2.10　BrakeSpeed_9.imp（実装）

```
IMPLEMENTATION
    BrakeSpeed_9

REFINES
    BrakeSpeed_8

IMPORTS
    CalBrakeSpeed

SEES
    Parameter1, Parameter2, Parameter3

DEFINITIONS
    "types.def"

CONCRETE_VARIABLES
    grad, loc2

INITIALISATION
    grad := 0;
    loc2 := 0

PROPERTIES
    interval = c_interval &
    bdelay_e = delay_e &
    bdelay_n = delay_n &
    bdelay_w = delay_w &
    margin_e = cmargin_e &
    margin_n = cmargin_n &
    margin_w = cmargin_w &
    release = warning_release

OPERATIONS
    SetSpeed(ta, lo, gr) =
    VAR ii IN
        grad := gr;
        ii := ta - lo;
        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;

    sp <-- Emergencyspeed =
    IF loc2 <= cmargin_e THEN
        sp := 0
    ELSE
        sp <-- Emergency_Speed(loc2, grad)
    END;

    sp <-- Warningspeed =
    IF loc2 <= cmargin_w THEN
        sp := 0
    ELSE
        sp <-- Warning_Speed(loc2, grad)
    END;

    sp <-- Releasespeed =
    IF loc2 <= cmargin_w THEN
        sp := 0
    ELSE
        VAR xx IN
```

```
73              xx <-- Warning_Speed(loc2, grad);
74              IF xx >= warning_release THEN
75                  sp := xx - warning_release
76              ELSE
77                  sp := 0
78              END
79          END
80      END
81
82  END
```

D.3 CalBrakeSpeed マシン

D.3.1 CalBrakeSpeed.mch（抽象機械）

```
1   MACHINE
2       CalBrakeSpeed
3
4   INCLUDES
5       Distance
6
7   DEFINITIONS
8       "types.def";
9       LOCATION == (0..MaxDist)
10
11  SEES
12      Parameter1
13
14  PROMOTES
15      NormalDistance, EmergencyDistance
16
17  CONSTANTS
18      c_interval, delay_e, delay_n, delay_w , cmargin_n, cmargin_e, cmargin_w
19
20  PROPERTIES
21      c_interval : 0..16383 &
22      delay_e : 0..16383 &
23      delay_n : 0..16383 &
24      delay_w : 0..16383 &
25      c_interval + delay_e : 0..16383 &
26      c_interval + delay_n : 0..16383 &
27      c_interval + delay_w : 0..16383 &
28
29      delay_e <= delay_n &
30      delay_n <= delay_w &
31
32      cmargin_e : 0..8191 &
33      cmargin_n : 0..8191 &
34      cmargin_w : 0..8191 &
35
36      cmargin_n >= cmargin_e &
37      cmargin_w >= cmargin_n
38
39  ASSERTIONS
40      !sp.(sp : SPEED => sp * (c_interval + delay_e)/36000 <= sp * (c_interval + delay_n)/36000);
41      !sp.(sp : SPEED => sp * (c_interval + delay_n)/36000 <= sp * (c_interval + delay_w)/36000);
42
43      !lm.(lm : 0..32767 => lm * (c_interval + delay_e) / 36000 <= 32767);
44      !lm.(lm : 0..32767 => lm * (c_interval + delay_n) / 36000 <= 32767);
45      !lm.(lm : 0..32767 => lm * (c_interval + delay_w) / 36000 <= 32767);
46
47      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
48          (lm1 <= lm2 => lm1 * (c_interval + delay_e) / 36000 <= lm2 * (c_interval + delay_e) / 36000));
49      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
50          (lm1 <= lm2 => lm1 * (c_interval + delay_n) / 36000 <= lm2 * (c_interval + delay_n) / 36000));
51      !(lm1, lm2).((lm1 : 0..32767 & lm2 : 0..32767) =>
52          (lm1 <= lm2 => lm1 * (c_interval + delay_w) / 36000 <= lm2 * (c_interval + delay_w) / 36000))
53
54  VARIABLES
55      normal_speed, emergency_speed, warning_speed
56
57  INVARIANT
58      emergency_speed : LOCATION -->(GRAD --> SPEED) &
59      !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
```

```
60          (loc0 <= cmargin_e => emergency_speed(loc0)(gr) = 0) &
61          (not(loc0 <= cmargin_e) => emergency_speed(loc0)(gr) =
62              max({sp | sp : SPEED &
63                sp * (c_interval + delay_e) / 36000 + distance_e(sp, gr) <= loc0 - cmargin_e})) &
64
65      normal_speed : LOCATION -->(GRAD --> SPEED) &
66      !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
67          (loc0 <= cmargin_n => normal_speed(loc0)(gr) = 0) &
68          (not(loc0 <= cmargin_n) => normal_speed(loc0)(gr) =
69              max({sp | sp : SPEED &
70                sp * (c_interval + delay_n) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_n})) &
71
72      warning_speed : LOCATION -->(GRAD --> SPEED) &
73      !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
74          (loc0 <= cmargin_w => (warning_speed(loc0))(gr) = 0) &
75          (not(loc0 <= cmargin_w) => (warning_speed(loc0))(gr) =
76              max({sp | sp : SPEED &
77                sp *(c_interval + delay_w) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_w}))
78
79  INITIALISATION
80      emergency_speed :(
81          emergency_speed : LOCATION -->(GRAD --> SPEED) &
82          !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
83              (loc0 <= cmargin_e => emergency_speed(loc0)(gr) = 0) &
84              (not(loc0 <= cmargin_e) => emergency_speed(loc0)(gr) =
85                  max({sp | sp : SPEED &
86                    sp * (c_interval + delay_e) / 36000 + distance_e(sp, gr) <= loc0 - cmargin_e})))) ||
87
88      normal_speed :(
89          normal_speed : LOCATION -->(GRAD --> SPEED) &
90          !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
91          (loc0 <= cmargin_n => normal_speed(loc0)(gr) = 0) &
92          (not(loc0 <= cmargin_n) => normal_speed(loc0)(gr) =
93              max({sp | sp : SPEED &
94                sp *(c_interval + delay_n) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_n})))) ||
95
96      warning_speed :(
97          warning_speed : LOCATION -->(GRAD --> SPEED) &
98          !(loc0, gr).(loc0 : LOCATION & gr : GRAD =>
99              (loc0 <= cmargin_w => (warning_speed(loc0))(gr) = 0) &
100             (not(loc0 <= cmargin_w) => (warning_speed(loc0))(gr) =
101                 max({sp | sp : SPEED &
102                   sp *(c_interval + delay_w) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_w}))))
103
104 OPERATIONS
105     sp <-- Normal_Speed(loc0, gr) =
106     PRE
107         loc0 : LOCATION &
108         gr : GRAD
109     THEN
110         sp := normal_speed(loc0)(gr)
111     END;
112
113     sp <-- Emergency_Speed(loc0, gr) =
114     PRE
115         loc0 : LOCATION &
116         gr : GRAD
117     THEN
118         sp := emergency_speed(loc0)(gr)
119     END;
120
121     sp <-- Warning_Speed(loc0, gr) =
122     PRE
123         loc0 : LOCATION &
124         gr : GRAD
125     THEN
126         sp := warning_speed(loc0)(gr)
127     END
128
129 END
```

D.3.2　CalBrakeSpeed_1.ref（詳細化 1）

```
1   REFINEMENT
2       CalBrakeSpeed_1
```

```
3
4   REFINES
5       CalBrakeSpeed
6
7   INCLUDES
8       Distance
9
10  DEFINITIONS
11      "types.def";
12      LOCATION == (0..MaxDist)
13
14  SEES
15      Parameter1
16
17  PROMOTES
18      NormalDistance, EmergencyDistance
19
20  VARIABLES
21      emergency_speed2, normal_speed2, warning_speed2
22
23  INVARIANT
24      emergency_speed2 : LOCATION +->(GRAD --> SPEED) &
25      emergency_speed2 = ((cmargin_e+1)..MaxDist)  <| emergency_speed &
26      normal_speed2 : LOCATION +-> (GRAD --> SPEED) &
27      normal_speed2 = ((cmargin_n+1)..MaxDist)  <| normal_speed &
28      warning_speed2 : LOCATION +-> (GRAD --> SPEED) &
29      warning_speed2 = ((cmargin_w+1)..MaxDist)  <| warning_speed
30
31  INITIALISATION
32      emergency_speed2 :(
33          emergency_speed2 : LOCATION +->(GRAD --> SPEED) &
34          dom(emergency_speed2) = (cmargin_e+1)..MaxDist &
35          !(loc0, gr).(loc0 : (cmargin_e+1)..MaxDist & gr : GRAD =>
36              emergency_speed2(loc0)(gr) =
37                  max({sp | sp : SPEED &
38                    sp *(c_interval + delay_e) / 36000 + distance_e(sp, gr) <= loc0 - cmargin_e})) ||
39
40      normal_speed2 :(
41          normal_speed2 : LOCATION +->(GRAD --> SPEED) &
42          dom(normal_speed2) = (cmargin_n+1)..MaxDist &
43          !(loc0, gr).(loc0 : (cmargin_n+1)..MaxDist & gr : GRAD =>
44               normal_speed2(loc0)(gr) =
45              max({sp | sp : SPEED &
46                sp *(c_interval + delay_n) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_n})) ||
47
48      warning_speed2 :(
49          warning_speed2 : LOCATION +->(GRAD --> SPEED) &
50          dom(warning_speed2) = (cmargin_w+1)..MaxDist &
51          !(loc0, gr).(loc0 : (cmargin_w+1)..MaxDist & gr : GRAD =>
52          warning_speed2(loc0)(gr) =
53              max({sp | sp : SPEED &
54                sp *(c_interval + delay_w) / 36000 + distance_n(sp, gr) <= loc0 - cmargin_w}))
55
56  OPERATIONS
57      sp <-- Emergency_Speed(loc0, gr) =
58      PRE
59          loc0 : LOCATION &
60          gr : GRAD
61      THEN
62          IF loc0 <= cmargin_e THEN
63              sp := 0
64          ELSE
65              sp := emergency_speed2(loc0)(gr)
66          END
67      END;
68
69      sp <-- Normal_Speed(loc0, gr) =
70      PRE
71          loc0 : LOCATION &
72          gr : GRAD
73      THEN
74          IF loc0 <= cmargin_n THEN
75              sp := 0
76          ELSE
77              sp := normal_speed2(loc0)(gr)
78          END
79      END;
```

```
80
81      sp <-- Warning_Speed(loc0, gr) =
82      PRE
83          loc0 : LOCATION &
84          gr : GRAD
85      THEN
86          IF loc0 <= cmargin_w THEN
87              sp := 0
88          ELSE
89              sp := warning_speed2(loc0)(gr)
90          END
91      END
92
93  END
```

D.3.3　CalBrakeSpeed_2.ref（詳細化 2）

```
1   REFINEMENT
2       CalBrakeSpeed_2
3
4   REFINES
5       CalBrakeSpeed_1
6
7   DEFINITIONS
8       "types.def";
9       LOCATION == (0..MaxDist)
10
11  INCLUDES
12      Distance
13
14  SEES
15      Parameter1
16
17  PROMOTES
18      NormalDistance, EmergencyDistance
19
20  OPERATIONS
21      sp <-- Emergency_Speed(loc0, gr) =
22      PRE
23          loc0 : LOCATION &
24          gr : GRAD
25      THEN
26          IF loc0 <= cmargin_e THEN
27              sp := 0
28          ELSE
29              sp := max({sp1 | sp1 : SPEED &
30                sp1 *(c_interval + delay_e) / 36000 + distance_e(sp1, gr) <= loc0 - cmargin_e})
31          END
32      END;
33
34      sp <-- Normal_Speed(loc0, gr) =
35      PRE
36          loc0 : LOCATION &
37          gr : GRAD
38      THEN
39          IF loc0 <= cmargin_n THEN
40              sp := 0
41          ELSE
42              sp := max({sp1 | sp1 : SPEED &
43                sp1 *(c_interval + delay_n) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_n})
44          END
45      END;
46
47
48      sp <-- Warning_Speed(loc0, gr) =
49      PRE
50          loc0 : LOCATION &
51          gr : GRAD
52      THEN
53          IF loc0 <= cmargin_w THEN
54              sp := 0
55          ELSE
56              sp := max({sp1 | sp1 : SPEED &
57                sp1 *(c_interval + delay_w) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_w})
58          END
```

```
59      END
60
61  END
```

D.3.4 CalBrakeSpeed_3.imp（実装）

```
1   IMPLEMENTATION
2       CalBrakeSpeed_3
3
4   REFINES
5       CalBrakeSpeed_2
6
7   DEFINITIONS
8       "types.def"
9
10  IMPORTS
11      Distance
12
13  SEES
14      Parameter1, Parameter3
15
16  PROMOTES
17      NormalDistance, EmergencyDistance
18
19  VALUES
20      c_interval = cinterval;
21      delay_e = cdelay_e;
22      delay_n = cdelay_n;
23      delay_w = cdelay_w;
24      cmargin_e = c_margin_e;
25      cmargin_n = c_margin_n;
26      cmargin_w = c_margin_w
27
28  OPERATIONS
29      sp <-- Emergency_Speed(loc0, gr) =
30      IF loc0 <= cmargin_e
31      THEN
32          sp := 0
33      ELSE
34          VAR ll, hh, ii IN
35              ll := 0;
36              hh := MaxSpeed;
37              ii := 0;
38              sp := 0;
39              WHILE ll /= hh DO
40                  sp := (ll + 1 + hh) / 2;
41                  ii := 0;
42                  ii <-- EmergencyDistance(sp, gr);
43                  ii := sp * (c_interval + delay_e) / 36000 + ii - loc0 + cmargin_e;
44                  IF ii <= 0 THEN
45                      ll := sp
46                  ELSE
47                      sp := sp - 1;
48                      hh := sp
49                  END
50              INVARIANT
51                  ii : INT &
52                  ll : SPEED &
53                  hh : SPEED &
54                  sp : ll..hh &
55                  ll : {sp1 | sp1 : SPEED &
56                          sp1 * (c_interval + delay_e) / 36000 + distance_e(sp1, gr) <= loc0 - cmargin_e} &
57                  (hh + 1) /: {sp1 | sp1 : SPEED &
58                          sp1 * (c_interval + delay_e) / 36000 + distance_e(sp1, gr) <= loc0 - cmargin_e}
59              VARIANT
60                  hh - ll
61              END
62          END
63      END;
64
65      sp <-- Normal_Speed(loc0, gr) =
66      IF loc0 <= cmargin_n
67      THEN
68          sp := 0
69      ELSE
```

```
70          VAR ll, hh, ii IN
71              ll := 0;
72              hh := MaxSpeed;
73              ii := 0;
74              sp := 0;
75              WHILE ll /= hh DO
76                  sp := (ll + 1 + hh) / 2;
77                  ii := 0;
78                  ii <-- NormalDistance(sp, gr);
79                  ii := sp * (c_interval + delay_n) / 36000 + ii - loc0 + cmargin_n;
80                  IF ii <= 0 THEN
81                      ll := sp
82                  ELSE
83                      sp := sp - 1;
84                      hh := sp
85                  END
86              INVARIANT
87                  ii : INT &
88                  ll : SPEED &
89                  hh : SPEED &
90                  sp : ll..hh &
91                  ll : {sp1 | sp1 : SPEED &
92                          sp1 * (c_interval + delay_n) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_n} &
93                  (hh + 1) /:    {sp1 | sp1 : SPEED &
94                          sp1 * (c_interval + delay_n) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_n}
95              VARIANT
96                  hh - ll
97              END
98          END
99      END;
100
101     sp <-- Warning_Speed(loc0, gr) =
102     IF loc0 <= cmargin_w
103     THEN
104         sp := 0
105     ELSE
106         VAR ll, hh, ii IN
107             ll := 0;
108             hh := MaxSpeed;
109             ii := 0;
110             sp := 0;
111             WHILE ll /= hh DO
112                 sp := (ll + 1 + hh) / 2;
113                 ii := 0;
114                 ii <-- NormalDistance(sp, gr);
115                 ii := sp * (c_interval + delay_w) / 36000 + ii - loc0 + cmargin_w;
116                 IF ii <= 0 THEN
117                     ll := sp
118                 ELSE
119                     sp := sp - 1;
120                     hh := sp
121                 END
122             INVARIANT
123                 ii : INT &
124                 ll : SPEED &
125                 hh : SPEED &
126                 sp : ll..hh &
127                 ll : {sp1 | sp1 : SPEED &
128                         sp1 * (c_interval + delay_w) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_w} &
129                 (hh + 1) /: {sp1 | sp1 : SPEED &
130                         sp1 * (c_interval + delay_w) / 36000 + distance_n(sp1, gr) <= loc0 - cmargin_w}
131             VARIANT
132                 hh - ll
133             END
134         END
135     END
136
137 END
```

D.4 Distance マシン

D.4.1 Distance.mch（抽象機械）

```
MACHINE
    Distance

SEES
    Parameter1

DEFINITIONS
    "types.def";

    inv_distance_e ==

        distance_e : SPEED * GRAD --> 0..(MaxSpeed * MaxSpeed) &
        !gr.(gr : GRAD  => distance_e(0, gr) = 0) &
        !(sp1, sp2, gr).(sp1 : SPEED & sp2 : SPEED & gr : GRAD =>
               (sp1 >= sp2 => distance_e(sp1, gr) >= distance_e(sp2, gr))) &
        !(sp, gr1, gr2).(sp : SPEED & gr1: GRAD & gr2 : GRAD =>
              (gr1 >= gr2 => distance_e(sp, gr1) >= distance_e(sp, gr2)));

    inv_distance_n ==

        distance_n : SPEED * GRAD --> 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_n(sp, gr2)));

    inv_distancene ==
        !(sp, gr).(sp : SPEED & gr : GRAD => distance_n(sp, gr) >= distance_e(sp, gr))

VARIABLES
    distance_e, distance_n

INVARIANT
    inv_distance_e &
    inv_distance_n &
    inv_distancene

INITIALISATION
    distance_e, distance_n :(inv_distance_e & inv_distance_n & inv_distancene)

OPERATIONS
    dn <-- NormalDistance(sp, gr) =
    PRE
        sp : SPEED &
        gr : GRAD
    THEN
        dn :=  distance_n(sp, gr)
    END;

    de <-- EmergencyDistance(sp, gr) =
    PRE
        sp : SPEED &
        gr : GRAD
    THEN
        de :=  distance_e(sp, gr)
    END

END
```

D.4.2 Distance_1.ref（詳細化 1）

```
REFINEMENT
    Distance_1

REFINES
```

```
    Distance

SEES
    Parameter1

DEFINITIONS
    "types.def"

CONSTANTS
    N_decl, E_decl

PROPERTIES
    N_decl: 1..255 &
    E_decl: 1..255 &
    N_decl <= E_decl &
    N_decl * 72 > MaxGrad * Gr1 * 72 / Gr2

VARIABLES
    distance_n, distance_e

INVARIANT
    !(sp, gr).(sp : SPEED & gr : GRAD =>
        distance_n(sp, gr) = sp * sp / (N_decl * 72 - gr * Gr1 * 72 / Gr2)) &

    !(sp, gr).(sp : SPEED & gr : GRAD =>
        distance_e(sp, gr) = sp * sp / (E_decl * 72 - gr * Gr1 * 72 / Gr2))

INITIALISATION
    ASSERT
        !(xx, yy).(xx : NAT & yy : NAT1 => xx/yy = max({tt | tt : NAT & yy * tt <= xx})) &
        !(xx, yy, zz).(xx : NAT & yy : NAT1 & zz : NAT1 => (yy <= zz => xx/zz <= xx/yy))
    THEN
        ASSERT
            !(xx, yy, zz).(xx : NAT & yy : NAT & zz : NAT1 => (xx <= yy => xx/zz <= yy/zz))
        THEN
            ASSERT
                !gr.(gr : GRAD => (N_decl * 72 - gr * Gr1 * 72 / Gr2 > 0)) &
                !gr.(gr : GRAD => (E_decl * 72 - gr * Gr1 * 72 / Gr2 > 0))
            THEN
                distance_n := %(sp, gr).(sp: SPEED & gr: GRAD
                                | (sp * sp) / (N_decl * 72 - gr * Gr1 * 72 / Gr2)) ||
                distance_e := %(sp, gr).(sp: SPEED & gr: GRAD
                                | (sp * sp) / (E_decl * 72 - gr * Gr1 * 72 / Gr2))
            END
        END
    END

OPERATIONS
    dn <-- NormalDistance(sp, gr) =
        PRE
            sp : SPEED &
            gr : GRAD
        THEN
            dn :=  distance_n(sp, gr)
        END;

    de <-- EmergencyDistance(sp, gr) =
        PRE
            sp : SPEED &
            gr : GRAD
        THEN
            de :=  distance_e(sp, gr)
        END

END
```

D.4.3　Distance_2.imp（実装）

```
IMPLEMENTATION
    Distance_2

REFINES
    Distance_1

SEES
```

```
8       Parameter1
9
10  DEFINITIONS
11      "types.def"
12
13  VALUES
14      N_decl = n_decl;
15      E_decl = e_decl
16
17  OPERATIONS
18      dn <-- NormalDistance(sp, gr) =
19      VAR yy IN
20          IF gr < 0 THEN yy := 0
21          ELSE yy := gr * Gr1 * 72 / Gr2 END;
22          IF yy < N_decl *72 THEN
23              dn := (sp * sp) / (N_decl  * 72 - yy)
24          ELSE
25              dn := MaxSpeed * MaxSpeed
26          END
27      END;
28
29      de <-- EmergencyDistance(sp, gr) =
30      VAR yy IN
31          IF gr < 0 THEN yy := 0
32          ELSE yy := gr * Gr1 * 72 / Gr2 END;
33          IF yy < E_decl * 72 THEN
34              de := (sp * sp) / (E_decl * 72 - yy)
35          ELSE
36              de := MaxSpeed * MaxSpeed
37          END
38      END
39
40  END
```

索引