修 士 論 文

代数仕様言語 CafeOBJ による鉄道信号システムの記述

指導教官　二木 厚吉 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻
言語設計学講座

清野 貴博

1999 年 2 月 15 日

要 旨

鉄道信号は大規模な分散型のシステムのうち、最も成功している事例の一つである。鉄道信号において、重要な概念は「閉そく (Block system) 」と「連動 (Interlocking) 」である。閉そくとは、線路をいくつかの区間に区切り、その区間を一列車に占有させるという考え方である。連動とは、個々の装置の間の関係を決めるもので、ある装置を取り扱った時に、別の装置を取り扱うことができないようにする「鎖錠関係」を持たせ、それを保ちながら動作することを言う。

本研究では、我が国の鉄道で使用されている鉄道信号システムを対象とし、そこで用いられている閉そくの概念とその実現法について記述した。

複線区間の信号システムは、レール、列車、軌道回路 (Track Circuit) 、信号機の各オブジェクトを合成して記述した。複線区間においては、上り線、下り線のように列車進行方向が固定のため、列車同士の追突を防止することを考える。まず、レールを連続した閉そく区間に分割し、各区間にはその区間に列車がいるかどうかを検出する軌道回路を設置する。各区間の始端には信号機を設置し、その区間に列車がいれば停止信号を、いなければ進行信号を現示し、後続の列車に区間の状況を知らせる。また、今日では停止信号と進行信号だけではなく、高速、高密度運転に対応するため、停止信号を予告する注意信号なども使用されているため、それらの信号を含めた閉そくシステムを記述し、線路の長さや閉そく区間の数などに依存せず、閉そくが保たれていることを余帰納法により証明した。

単線区間の信号システムでは、一本の線路上を双方向に列車が走るため、無秩序に列車が走ると正面衝突の危険がある。そこで、複線区間の設備の他に、列車交換のための設備と、どちらの方向に列車を運転するのかを決める方向てこが必要になる。そこで、これらのオブジェクトを複線区間の信号システムに加え、正面衝突が起こらないこと、閉そくが保たれていることを、同様に余帰納法により証明した。

これらの仕様は限定的ではあるが、具体的な線路配置に依存しないように記述した。本研究では、これら証明済みの信号付き線路コンポーネントを組み合わせて、鉄道システムを作り上げるための足がかりを示した。

目 次

第 1章

序論

1.1　背景

コンピュータの高機能化、低価格化などのハードウェア的な要因、ネットワークの普及によるユーザの要求の高度化などのソフトウェア的な要因から、並列分散型のアーキテクチャを持つシステムが望まれている。しかし、そのようなシステムは設計が難しく、その挙動を設計段階において完全に予測することも困難である。このような中で、並列分散型のシステムを科学的に設計するための方法として、形式手法 (Formal method) が有望視されている。

形式手法に基づいた記述した仕様を形式仕様と呼ぶ。仕様は自然語で書かれることが多かったが、形式仕様は自然言語による仕様と比較して、無矛盾性、完全性、非曖昧性、明晰性に優れているとされ、新たなコミュニケーションメディアとしても有効性が認知されつつある。

このような中で、これまで例題程度しか形式仕様の適用事例のなかった並列分散型のシステムに対して、中〜大規模なシステムに適用を試みる事例が現れ、形式仕様の有効性を実証しようという研究が盛んになりつつある。

1.2　大規模並列分散システムの成功事例: 鉄道

鉄道は大規模並列分散システムの成功事例の一つである。今日、鉄道はその安全性、定時性などが高く評価され、社会基盤の一つとして無くてはならない地位を築いている。

鉄道の安全を確保する上で一翼を担うものが、鉄道信号である。鉄道信号は国によって、その考え方を異にするが、多くは並列分散型のアーキテクチャで構成されている。そ

の根幹をなす概念は「閉そく (Block system) 」である。閉そくとは、線路を一定の区間に区切り、その区間を一列車に占有させるということである。これは鉄のレールと鉄の車輪では摩擦が小さく、見通し距離では停止できないという鉄道特有の事情を前提としているからであり、自動車の信号とは全く異なるものである。これは、線路とは閉そく区間が連なったものとして捉えることができることを意味している。

鉄道信号において、連動 (Interlocking) も重要な概念である。鉄道信号では、現示される信号が機械的に決定される自動信号と、人間の介在を必要とする半自動の信号の二つがある。しかし、人間は間違いを犯すものであり、謝った操作や一貫性のない操作は信号系の混乱につながり、それは事故の原因となる。例えば、列車が通過中の分岐器を転換したら、列車は脱線するだろう。この例から分かるように、使用中の装置は危険な操作ができないように鎖錠しておく必要がある。このような関係を規定することが連動である。

並列分散システムの成功事例である鉄道を、形式仕様を用いて記述することによって得られる知見は、鉄道以外の並列分散システムに形式仕様を適用する場合においても応用できると考えられる。本研究では、長く鉄道の安全を守ってきた「閉そく」と「連動」の概念に焦点を当て、これらを数学的基盤を持つ形式仕様によって記述し、その安全性を検証する。

1.3 本書の構成

本研究では、鉄道信号システムを形式仕様を用いて記述し、その仕様を用いた科学的検証によって、信号システムの安全性について論じる。対象とする鉄道は、日本の一般的な鉄道である。

2 章 本研究において採用した仕様記述言語 CafeOBJ について、その背景となる数学的基盤である隠蔽代数や、CafeOBJ における仕様記述について触れる。

3 章 CafeOBJ を用いて、原始的な鉄道システム Bare-Railway を記述する。Bare-Railway は保安装置が全くない危険な鉄道システムである。この仕様に次章以降で様々な保安装置を追加する。

4 章 複線区間における信号システムについて仕様を記述し、その安全性を検証する。複線区間では、列車の進行方向が決められているため、正面衝突は起こらないが、見通し距離で停止できない列車は追突の危険がある。そこで、追突を防ぐための信号システムについて考察する。

5章 単線区間における信号システムについて仕様を記述し、その安全性を検証する。単線区間では、列車が双方向に走るため、追突に加えて正面衝突の危険がある。追突の防止法は複線区間と同様であるが、正面衝突の防止には新たな保安装置が必要となる。その方法は種々であるが、最も近代的な方法である「方向てこ」を取り上げ、これによる対向列車抑止の方法について考察する。

6章 本研究で得られた知見をまとめ、関連研究と本研究との関係や今後の課題について触れる。

第 2章

代数仕様言語 CafeOBJ

2.1 概要

CafeOBJ は代数を基盤とする形式仕様 (Formal specification) を記述するための言語である。このような言語を代数仕様言語 (Algebraic specification language) と呼ぶ。CafeOBJ は代数仕様言語 OBJ を祖とし、強力なモジュールシステム、通常の言語の型に相当する順序ソート (Ordered Sort) 、実装をブラックボックスとしてオブジェクトを記述する隠蔽代数 (Hidden Algebra) のサポート、項書換えによる仕様の実行など、様々な特色を備えている先進的な言語である [1] 。

2.2 隠蔽代数と振舞仕様

CafeOBJ では、オブジェクトを Joseph Goguen と Grant Malcom によって提唱された隠蔽代数 [4] によって扱う。隠蔽代数は、カプセル化されたオブジェクトの状態を扱うことができ、オブジェクトの状態は隠蔽ソート (Hidden sort) によって表現される。

隠蔽ソート上には、オブジェクトの状態を表現する操作演算 (action) が定義される。オブジェクトは操作演算が行われれる度に、自身の状態 (振舞い) を変化させる。操作演算は引数として任意の数の可視ソートと隠蔽ソートを持つ。隠蔽ソート上には、オブジェクトの状態を可視ソートとして取り出すことができる観測演算 (observation) も定義する [1] 。操作演算の列によって表現されたオブジェクトの状態を可視ソートにマップするための規則は、等式によって与える。このようにして、隠蔽代数によって記述された仕様を振舞仕様と呼ぶ。

[1] 文献によっては操作演算は method, 観測演算は attribute となっていることもある

以下に例として、ON, OFF の二つの状態を持つようなオブジェクト「スイッチ」の例を示す。

```
mod* SWITCH {
  *[ Switch ]*
  bop status : Switch -> Bool
  bop on     : Switch -> Switch
  bop off    : Switch -> Switch

  var SW : Switch
  eq status (on (SW)) = true .
  eq status (off (SW)) = false .
}
```

1 行目はモジュール SWITCH の定義を開始するための宣言である。2 行目の *[ ]* に囲まれた部分は、隠蔽ソート Switch の宣言である。ソート Switch 上の演算として、以下の三つを定義する。演算 on と off はスイッチの状態を変化させる操作演算である。操作演算 on, off は引数として隠蔽ソート Switch を取り、演算の結果として Switch を返す。status は観測演算であり、スイッチの状態を可視ソート Bool として返すことを宣言している。ソート Bool は CafeOBJ 組み込みのソートであり、true と false の二値とその上の演算が定義されている。

操作演算と観測演算は、オブジェクトとその外界のインターフェースを定義していると見ることができる。一方、操作演算によってオブジェクトの状態が変わり、観測演算の結果として得られる状態も変化するというような、オブジェクトの振舞いは、観測文脈 (visible context) によって定義される。観測文脈とは、オブジェクトに対して何回か操作演算を行った後、観測演算を適用し、オブジェクトの状態を知るような演算の列である。例えば、

1. status (on (off (off (on ...)))

2. status (off (on (off (on (on (...))))))

のような演算の列は観測文脈である。

オブジェクトの振舞いは、上記の仕様では、等式 eq によって与えられている。var とは変数の宣言であり、ソートが Switch であるような項すべてを指している。一つ目の eq は、status (on ...) で始まるような観測文脈は true であり、二つ目では、status (off ...) で始まるような観測文脈は false であることを定義している。

このようにして与えられた仕様は、実際に動かして見ることができる。CafeOBJ では等式を、左辺から右辺への書換え規則と見なし、項書換えに実行することができる。例えば前述の観測文脈 1 をこの等式を使って書き換えると、一つ目の等式が書き換え規則として適用され、true が返ってくる。

2.3 振舞等価と余帰納法

二つのオブジェクトに同じ操作演算を適用した時に、それらが同じ振舞いを持つことを振舞等価 (Behavioural equivalence) と言う。振舞等価性 $\equiv$ は以下のように定義される [4] 。

- when $s \in V$:
  $a \equiv a'$ iff $a = a'$
- when $s \in H$:
  $a \equiv a'$ iff $c(a) = c(a')$ for all $v \in V$ and for all visible contexts c

a, a' は観測演算、V は可視ソートの集合、H は隠蔽ソートの集合である。

振舞等価性を操作演算の列 (context) に関する帰納法によって証明する方法を context induction [6] と呼ぶ。しかし、context induction は大きな仕様では証明が複雑になることが知られており、代わりに余帰納法 (coinduction) が用いられる。振舞等価は最大の隠蔽合同であるという定理 [4] から、余帰納法によって仕様が振舞等価であることは、次の手順によって仕様が隠蔽合同であることを示せばよい [4] 。

1. 隠蔽合同関係 R を定義する。
2. R がすべての操作演算に対して隠蔽合同であることを示す。

以下では、前出の仕様 SWITCH が隠蔽合同であることを示す。

```
open SWITCH
op _R_ : Switch Switch -> Bool .
```

隠蔽合同関係 R を定義する。

```
vars SW1 SW2 : Switch
eq SW1 R SW2 = status (SW1) == status (SW2) .
```

仮定: sw1 と sw2 は隠蔽合同である。

```
ops sw1 sw2 : -> Switch .
eq status (sw1) = status (sw2) .
```

操作演算 on, off の適用後も、二つのオブジェクトが隠蔽合同であることを示す。

```
red status (on (sw1)) R status (on (sw2)) .
red status (off (sw1)) R status (off (sw2)) .
```

CafeOBJ の処理系は、基本的な仕様については、自動的に仕様が隠蔽合同の性質を満たしていることを証明することができ、証明できた場合には隠蔽合同関係 R に基づく振舞等価関係を示す演算子 =*= を、仕様に追加する。

2.4 射影演算

隠蔽代数によって記述されたオブジェクトをいくつか集めて合成し、より大きなオブジェクトを作成することができる。これには、合成後のオブジェクトの状態空間を、合成前のオブジェクトの状態空間に射影する、射影演算 (Projection Operator) [2] を用いる。

以下に、射影演算を用いた合成オブジェクトの例として、前出の SWITCH オブジェクト二つからなるようなオブジェクト TWO-SW の仕様を示す。

```
mod* TWO-SW {
  pr (SWITCH)
  *[ Two-Switch ]*
  bop sw1-on  : Two-Switch -> Two-Switch
  bop sw1-off : Two-Switch -> Two-Switch
  bop sw2-on  : Two-Switch -> Two-Switch
  bop sw2-off : Two-Switch -> Two-Switch
  bop sw1     : Two-Switch -> Switch
  bop sw2     : Two-Switch -> Switch

  var SW : Two-Switch
  eq sw1 (sw1-on (SW)) = on (sw1 (SW)) .
  eq sw1 (sw1-off (SW)) = off (sw1 (SW)) .
  eq sw1 (sw2-on (SW)) = sw1 (SW) .
  eq sw1 (sw2-off (SW)) = sw1 (SW) .
```

```
  eq sw2 (sw1-on (SW)) = sw2 (SW) .
  eq sw2 (sw1-off (SW)) = sw2 (SW) .
  eq sw2 (sw2-on (SW)) = on (sw2 (SW)) .
  eq sw2 (sw2-off (SW)) = off (sw2 (SW)) .
}
```

二行目の pr は仕様の輸入 (import) であり、モジュール SWITCH で定義された仕様を、TWO-SW でも利用できるようにすることを宣言している。次に、合成後のオブジェクトの隠蔽ソート Two-Switch を宣言する。ここでは、このオブジェクトは、二つのスイッチから成っており、それらを自由に操作することができる。スイッチ 1 を動かす操作演算が sw1-on, sw1-off であり、スイッチ 2 の操作演算が sw2-on, sw2-off である。

次に射影演算として、二つのスイッチの状態空間から、スイッチ 1 の状態空間を取り出す演算 sw1 と、スイッチ 2 の状態空間を取り出す演算 sw2 を定義する。これらの演算の意味を等式で与える。これらのスイッチは全く独立に動いており、互いに影響を与えることはない。隠蔽ソート Two-Switch 上の操作演算 sw1-on, sw1-off を sw1 の状態空間に、sw2-on, sw2-off を sw2 の状態空間にマップするように切り分けている。

次に、スイッチ 1 は、スイッチ 2 が ON になっている時にだけ操作できるようなオブジェクト TWO-SW+ の仕様を示す。

```
mod* TWO-SW+ {
  pr (SWITCH)
  *[ Two-Switch ]*
  bop sw1-on  : Two-Switch -> Two-Switch
  bop sw1-off : Two-Switch -> Two-Switch
  bop sw2-on  : Two-Switch -> Two-Switch
  bop sw2-off : Two-Switch -> Two-Switch
  bop sw1     : Two-Switch -> Switch
  bop sw2     : Two-Switch -> Switch

  var SW : Two-Switch
  ceq sw1 (sw1-on (SW)) = sw1 (SW)
      if  status (sw2 (SW)) == false .
  ceq sw1 (sw1-on (SW)) = on (sw1 (SW))
      if  status (sw2 (SW)) == true .
  ceq sw1 (sw1-off (SW)) = sw1 (SW)
      if  status (sw2 (SW)) == false .
  ceq sw1 (sw1-off (SW)) = off (sw1 (SW))
      if  status (sw2 (SW)) == true .
```

```
  eq sw2 (sw1-on (SW)) = sw2 (SW) .
  eq sw2 (sw1-off (SW)) = sw2 (SW) .
  eq sw2 (sw2-on (SW)) = on (sw2 (SW)) .
  eq sw2 (sw2-off (SW)) = off (sw2 (SW)) .
}
```

このようなオブジェクトの間の関係は、射影演算によって定義される。スイッチ 2 は自由に動かすことができるので、オブジェクト TWO-SW の定義と同様である。スイッチ 1 はスイッチ 2 の状態に依存することになるので、条件付き等式 (ceq) によって、スイッチ 1 の状態空間への射影方法を変化させる。例えば、最初の条件付き等式では、スイッチ 2 の状態が false (OFF) ならば、Two-Switch 上の操作演算 sw1-on は、無視するように定義している。

上記の例では、簡単のために sw1, sw2 と区別したが、一般的にはオブジェクトには可視ソートによって ID を与えることもできる。

振舞等価であるオブジェクトを組み合わせて、射影演算によって定義されたオブジェクトは、合成前のオブジェクトの振舞等価性を利用できる。これによって、合成後のオブジェクトも振舞等価であることを示すことができる。

第 3章

原始的な鉄道システム

3.1 Bare-Railway の概要

本章では、レールとその上を走る列車だけから成るような、原始的な鉄道システムの仕様を示す。本仕様では、レールや列車は分散オブジェクトとして表現され、鉄道システム中に、任意の数のレールや列車が存在することができる。また、列車はレール上を自在に動くことができるが、どんな危険な行為も許されているので、様々な事故が起こり得る。

3.2 Rail オブジェクト

Rail オブジェクトは、レールのつながりを保持しているオブジェクトである。ある長さを持つ区間が、一つの Rail オブジェクトに相当する。個々の Rail オブジェクトは RailID を持つ。Rail オブジェクトのシグネチャを以下に示す。

```
*[ Rail ]*
op  _ _          : RailID RailID -> Rail
bop left-rail _  : Rail -> RailID
bop right-rail _ : Rail -> RailID
```

Rail オブジェクトは隣接する Rail オブジェクトの RailID を保持する。隣接するオブジェクトは便宜上、右、左という呼び方を用いる。観測演算 left-rail, right-rail は、それぞれ隣接する左側、右側の Rail オブジェクトの RailID を返す演算である。なお、Rail オブジェクトは静的なオブジェクトであり、操作演算を持っていない。

3.3 Train オブジェクト

Train オブジェクトは、列車に相当するオブジェクトである。Train オブジェクトは Rail オブジェクトを単位として移動し、Rail オブジェクト上での現在位置 (RailID) と進行方向 (ソート Direction 上の値: Left, Right) を保持する。Train オブジェクトのシグネチャを以下に示す。

```
*[ Train ]*
bop where? _        : Train -> RailID
bop which? _        : Train -> Direction
--
bop move _ _        : RailID Train -> Train
bop turn-left _     : Train -> Train
bop turn-right _    : Train -> Train
```

操作演算 move は引数として RailID を持ち、列車を任意の Rail オブジェクト上へ列車を移動させることができる。turn-left, turn-right は列車の進行方向を Left, Right のどちらかに変更することができる。Train オブジェクトは観測演算として、列車の Rail オブジェクト上の現在位置を知ることができる where? と、現在の進行方向を知ることができる which? を持っている。

3.4 Bare-Railway オブジェクト

Bare-Railway オブジェクトは、Rail オブジェクトと Train オブジェクトを、射影演算によるオブジェクト合成を利用して作成する。以下に Bare-Railway オブジェクトのシグネチャを示す。

```
*[ Bare-Railway ]*
bop move-train _ _    : TrainID Bare-Railway -> Bare-Railway
bop train-left _ _    : TrainID Bare-Railway -> Bare-Railway
bop train-right _ _   : TrainID Bare-Railway -> Bare-Railway
--
bop train _ _         : TrainID Bare-Railway -> Train
bop rail _ _          : RailID Bare-Railway -> Rail
```

操作演算 move-train は任意の TrainID を持つ列車を、その列車の進行方向へ Rail オブジェクト一つ分移動させることができる。また train-left, train-right によって、任意の列車の進行方向を変えることができる。

射影演算 train は、Bare-Railway オブジェクトを Train オブジェクトの状態空間へマップしている。射影演算 rail も同様である。これらを利用して、Bare-Railway を構成する個々のオブジェクトの状態を知ることができる。

第 4章

複線区間における自動閉そく

4.1　鉄道の保安概念

4.1.1　閉そくの概念とその実現法

鉄道は鉄のレールの上を鉄の車輪で走行するシステムである。それゆえ、レールと車輪の間の摩擦係数が小さく、列車の運転士が進路前方の危険を発見してブレーキをかけたとしても、容易に停止することはできない。昼夜を問わず、人が正しく視認できる距離は600m と言われており、これを基準にして、列車が最高速度から非常ブレーキをかけて停止するまでの制動距離を決めている。

しかし、実際の軌道上では、曲線区間や隧道などが存在するため、常に 600m 先まで見通せるわけではない。そこで、前方の様子を運転士に知らせるための信号設備が必要になる。その信号現示の基本的な概念が「閉そく (Block system) 」である。閉そくとは、線路をいくつかの区間に区切り、その一区間を一列車に占有させるという考え方である。線路から任意の一区間を取り出したとき、その区間に一列車しか存在していなければ、衝突が起こっていないことは明白である。

閉そくの考え方には、時間間隔法 (Time Interval System) と空間間隔法 (Space Interval System) の 2 つがあると言われている。

時間間隔法は、先行する列車が駅を通過した後、一定の時分をおいて列車を運転する方法である。この方法では、先行列車が何らかの事由により遅れを生じた場合に、追突の怖れがあるので、危険である。時間間隔法は簡便ではあるが、高速、高密度運転が行われる線区には適用が難しく、現在では使われていない。

空間間隔法は、先行列車と後続列車との間に一定の距離を保つという考え方である。こ

の方法では、先行する列車が現在走行中の区間を通過するまで、後続列車はその区間に進入することが許されない。この方法は、各列車の最高速度が違う場合や、先行列車に運転遅延が生じた場合にも閉そくが保たれることから、安全性が高く、現在、多くの鉄道で採用されている。

4.1.2 ルートシグナルとスピードシグナル

鉄道における信号には 2 種類の考え方がある。一つはイギリスで考案されたルートシグナル (Route signal) である。いずれの場合においても、信号は閉そくが完了したことを意味していることに変わりがない。

ルートシグナルでは、信号は進路毎に現示され、各信号は自らの防護範囲とする進路の開通を現示する。進路が開通している場合は進行信号 G を、そうでない場合は停止信号 R を現示する。我が国では、イギリスから鉄道技術を輸入したため、ルートシグナルを根幹に据えている。ルートシグナルでは、列車が進むべき進路が明確に示されるが、分岐器を通過する際の速度制限などは運転士の記憶に頼ることになる。

対して、アメリカで考案されたスピードシグナル (Speed Signal) では、各信号はその防護範囲において、列車が出してよい最高速度を現示する。例えば、列車が先行する列車が存在する区間に接近するにつれ、速度制限がかかる。この方法では、曲線区間や分岐器の通過などの際に速度制限が必要な場合にも、それらの情報も信号を通じて統一的に現示することができる。従って、運転士は、その路線の事情を全く知らなくても、信号に従って列車の速度を変えるだけで、安全に運行することができる。しかし、進路が分岐する場合には、列車が実際に進入してみるまで、正しい進路に入ったかどうかはわからない。

我が国の在来線は前述したようにルートシグナルを採用しているが、高速、高密度運転を実現するためスピードシグナルの概念を取り入れたものになっている。信号には進行信号・停止信号の他に、速度が制限されている減速信号 YG 、注意信号 Y 、警戒信号 YY などが使用されている [1] 。一般的には、進行信号・注意信号・停止信号 の 3 現示が使用されるが、列車の運転本数などに応じて、減速信号や警戒信号も使用した 4 現示や 5 現示が採用されている。

[1]制限速度は各会社によって異なる。例として、JR 各社の制限速度は減速信号が 65km/h 、注意信号が 45km/h 、警戒信号が 25km/h である。

4.1.3 追突の防止

本章では複線区間における閉そくを考えているので、一本の線路の上では、列車の進行方向が一定であると仮定する [2] 。この場合、線路の上を一定方向に列車が走っている時、列車 B は先行する列車 A に追突する怖れがあるので、これを防止する必要がある (図 4.1) 。

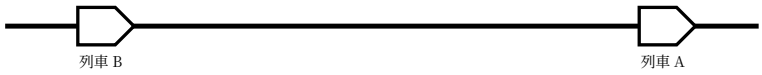

図 4.1: 追突が発生する場合

そこで、図 4.2 のように、線路をいくつかの区間に区切って、その区間の始端に信号機を設置する。

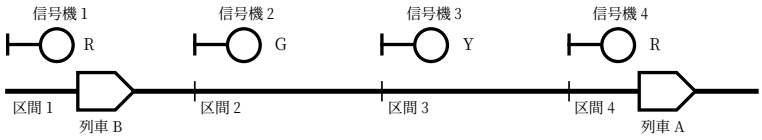

図 4.2: 追突防止の信号機

追突防止には、閉そくの概念を適用し、各区間に一列車しか入れないよう規制する。このような区間を閉そく区間という。図 4.2 の例では、信号機 1 は区間 1 の開通状況を現示する。区間 1 には既に列車 B が進入しているので、信号機 1 は後続の列車に対して、その区間に進入することができないという意味の停止信号 R を現示する。同様に、信号機 2 は区間 2 の開通を現示する。区間 2 には列車がおらず、信号機 2 は、その区間に進入することができるという意味の進行信号 G を現示する。

図 4.2 において、追突を防ぐためには、あらゆる場合において列車 B が区間 4 に進入しないことが必要であるが、列車 B の速度が高い時、列車 B が信号機 4 の停止信号を見てからブレーキをかけても、区間 4 に入るまでに停止できないかもしれない。そこで、区間 3 を防護範囲とする信号機 3 には、次の区間の信号機が停止信号 R を現示していることを予告する、注意信号 Y を現示させる。単に予告の意味だけでは不十分なので、前

[2]我が国の場合。アメリカなどでは複線区間においても、列車はそれぞれのレールの上を双方向に走る。

述の速度制限を付与している。列車 B は区間 3 に進入するまでに減速し、次の停止信号に備えることができる。

現実には、この議論は、閉そく区間の長さに依存する。十分に長い閉そく区間が確保できるならば、注意信号を現示する必要はないかもしれない。しかし、それには概算でも 2km 以上の閉そく区間が必要になり、2km 先の信号機は運転士には見えないし、一般的な列車の長さ 200m [3] と比較としても非常に大きく、長い区間を一列車に占有させると線路の利用効率が落ちる。逆に閉そく区間の長さを短くすると、線路の利用効率は上がるが、減速が間に合わないかもしれない。しかし、信号を減速信号、注意信号、警戒信号と段階的に現示することによって、短い閉そく区間でも安全に運転することができ、高速・高密度運転に対応している。

実際の路線では、閉そく区間の長さはまちまちで、信号機はその直前の区間の長さや運転計画によって適切なものが建植される。例えば、直前の区間が長ければ、進行信号、注意信号、停止信号の三現示信号機が建植され、直前の区間が短ければ、減速信号を加えた四現示信号機が建植されることになる。

4.2 自動閉そくとは

自動閉そくとは列車自体によって閉そくが行われる保安システムである。自動閉そくでは、停車場内外を問わず、線路を連続した閉そく区間に分割し、その区間を防護する信号機を各区間の始端に建植する。各信号機は、その区間に列車が存在する場合、自動的に停止信号を現示する。これによって各区間の閉そくが行われる。

自動閉そくにおける重要な前提条件として、すべての列車が停止信号までに確実に停止できることが挙げられる。従って、自動閉そくを実際の線区に適用する時は、線区の運転計画 (軌道設備、車両性能、計画運転線図、列車ダイヤ、最小運転時隔) によって各閉そく区間の長さを決め、列車の運転制御が確実に行われるかどうかを検討する必要があるとされている [7] [8]。

4.3 複線自動閉そくの設備

複線区間における自動閉そくの設備を図 4.3 に示す。各閉そく区間には軌道回路が設置され、その区間に列車が存在するかどうかを知ることができ、各信号機はこれを元に信号

[3] 20m 級車両 × 10 両で計算

を現示する。場内信号機及び出発信号機は構内配線によって必然的に決定され、閉そく信号機は列車運転計画から決定される。

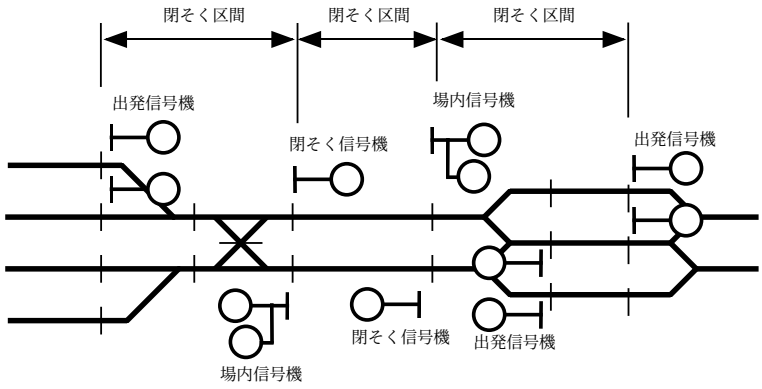

図 4.3: 複線区間における自動閉そくの設備

4.4 軌道回路

4.4.1 軌道回路の仕組み

自動閉そくを実現するためには、各閉そく区間に列車がいるかどうかを機械的に検知する必要がある。このための装置が軌道回路 (Track Circuit) である。

軌道回路は図 4.4 [4] のようにレールを絶縁継目で区切って構成する [5] 。この間のレールに信号電流を流す。すると、列車がいない間が信号電流によって、受電側の軌道リレーは打上し続ける。この区間に列車が進入すると、列車の車軸によって信号電流が短絡され、軌道リレーが落下する。また、レールにひびが入った場合や、停電の場合なども信号電流が流れなくなり、軌道リレーは落下する。軌道回路自体の異常は、信号系の異常となり危険であるので、フェールセーフのためにこれを停止信号としておく方が安全である。そこで、軌道リレーの落下側が停止信号の現示条件に、打上側が進行信号の現示条件として用いられている。

[4]出典: 白土義男, 信号システムの移り変わり, 鉄道ピクトリアル, 鉄道図書刊行会, Vol. 48, No. 7, 1998, pp. 23

[5]インピーダンスボンドに関する事項は、本稿では割愛する。

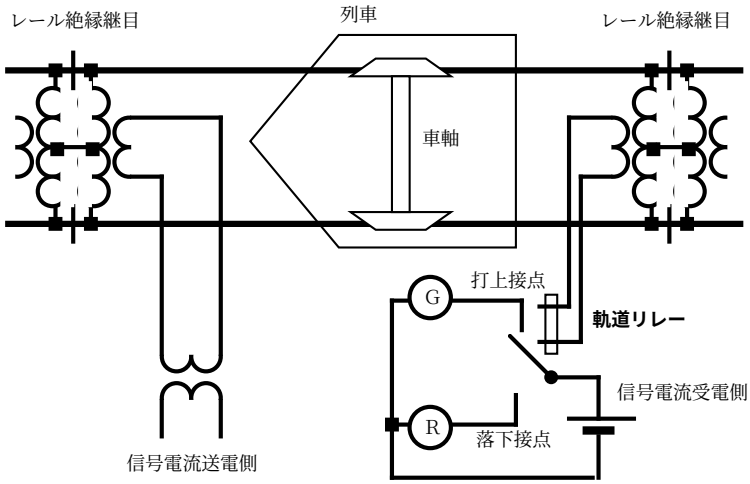

図 4.4: インピーダンスボンドと商用周波数軌道回路

4.4.2 軌道回路の仕様

Bare-Railway に軌道回路を敷設する。まずは軌道回路オブジェクトを作成する。以下に軌道回路のシグネチャを示す。

```
*[ TrackCircuit ]*
bop exist? _    : TrackCircuit -> Bool
bop in-train _  : TrackCircuit -> TrackCircuit
bop out-train _ : TrackCircuit -> TrackCircuit
```

軌道回路は、操作演算 in-train, out-train を持ち、これらは列車が進入した、列車が進出した時に適用される。軌道回路による列車の検出は、Bool 値を返す観測演算 exist? によって行う。exist? は、列車が存在している時は true を、そうでなければ false を返す。

次に、Bare-Railway オブジェクトと TrackCircuit オブジェクトの二つを合成し、軌道回路付きの鉄道システムとした Railway-TC を以下に示す。なお、軌道回路の ID は Rail オブジェクトと共用しており、Rail オブジェクト一つに対して一つの TrackCircuit オブジェクトが対応する。このことは、レールは無秩序に Rail オブジェクトに分割するのではなく、軌道回路を設置する単位で分割する必要があることを意味している。

```
*[ Railway-TC ]*
bop move-train _ _   : TrainID Railway-TC -> Railway-TC
bop train-left _ _   : TrainID Railway-TC -> Railway-TC
bop train-right _ _  : TrainID Railway-TC -> Railway-TC
```

```
--
bop railway _         : Railway-TC -> Bare-Railway
bop trackcircuit _ _ : RailID Railway-TC -> TrackCircuit
```

操作演算 move-train, train-left, train-right は Bare-Railway のものと変わりはない。Railway-TC は、そのコンポーネントである Bare-Railway を取り出すための射影演算 railway と、Railway-TC に含まれる任意 RailID を持つ TrackCircuit を取り出すための射影演算 trackcircuit を持つ。

以下に任意 RailID を持つ TrackCircuit を取り出すための射影演算 trackcircuit の一部を示す。

```
var R : RailID
var T : TrainID
var Rw : Railway-TC
ceq trackcircuit R (move-train T Rw) = trackcircuit R Rw
    if  where? (train T (railway Rw)) =/= R
    and where? (train T (railway (move-train T Rw))) =/= R .
ceq trackcircuit R (move-train T Rw) = in-train (trackcircuit R Rw)
    if  where? (train T (railway (move-train T Rw))) == R .
ceq trackcircuit R (move-train T Rw) = out-train (trackcircuit R Rw)
    if  where? (train T (railway Rw)) == R .
```

任意の RailID : R を持つ TrackCircuit に対する射影演算は、操作演算 move-train に対して、三通りに定義されている。このことは、move-train によって列車が動くことによって、TrackCircuit は三通りの動作をすることを意味している。

一つは、Train T の現在位置と RailID : R が一致する TrackCircuit の場合である。これは、TrackCircuit R 上に Train T が存在することを意味している。この Train T は move-train T Rw によって、TrackCircuit R の区間から進出する。そこで、TrackCircuit R には、列車の進出を意味する操作演算 out-train が適用される。これは、上記の射影演算に関する等式のうち、二つ目の条件付き等式で表現されている。

逆に、move-train T Rw の後の Train T の位置と RailID : R が一致する TrackCircuit の場合、列車が move-train T Rw によって、その区間に進入してくることを意味している。従って、列車の進入を意味する in-train が適用される (三つ目の条件付き等式) 。

これらに該当しない TrackCircuit R は、move-train T Rw によって動く列車に関係する軌道回路ではないので、操作演算は適用されない。

4.5 複線自動閉そく

4.5.1 閉そく区間の仕様

複線自動閉そくにおいて、閉そくの単位となる閉そく区間オブジェクトの仕様を以下に示す。

```
[ SectionID ]

*[ Section ]*
bop exist? _    : Section -> Bool
bop in-train _  : Section -> Section
bop out-train _ : Section -> Section
```

閉そく区間は、１つ以上の軌道回路が敷設された Rail オブジェクトに対して設定される。システムは任意の数の閉そく区間を持っており、個々の区間は SectionID によって識別される。

4.5.2 自動信号機の仕様

複線自動閉そくで使用する自動信号機の仕様を以下に示す。自動信号機は、信号機が現示する信号 SigValue 及び Signal オブジェクトで定義する。

```
[ SigValue ]
ops R Y G : -> SigValue

*[ Signal ]*
op  init-signal _ : List -> Signal
bop range _       : Signal -> List
bop watch _       : Signal -> SigValue
--
bop turn-R _      : Signal -> Signal
bop turn-Y _      : Signal -> Signal
bop turn-G _      : Signal -> Signal
```

信号機は、軌道回路を単位とする防護範囲を持っており、防護範囲内に列車が存在すれば R が現示される。そこで、Signal オブジェクトは、その防護範囲を示す、RailID のリストを静的に持っている。これは、閉そく信号機と軌道回路は一対一に対応するが、防護範囲に分岐器を含むような出発信号機や場内信号機は、複数の軌道回路を防護範囲と

して持つことがあるからである。Signal オブジェクトの観測演算はそのリストを取得する range と信号機の現示を取得する watch の二つである。操作演算は、信号機の現示を R/Y/G に変化させる turn-R / turn-Y / turn-G の三つである。

4.5.3 複線自動閉そくの仕様

軌道回路が敷設されている鉄道システム Railway-TC オブジェクトと、信号機オブジェクトを組み合わせて、複線自動閉そく Auto-Sys オブジェクトを定義する。

```
*[ Auto-Sys ]*

bop move-train _ _ : TrainID Auto-Sys -> Auto-Sys
--
bop railway _      : AutoSys -> Railway-TC
bop section _ _    : SectionID AutoSys -> Section
bop signal _ _     : SectionID AutoSys -> Signal
--
op  tcexpand _ _   : List AutoSys -> Bool
```

Auto-Sys では、操作演算は move-train だけに制限される。各列車の進行方向はあらかじめ決められており、勝手にそれを変えることは許されない。射影演算 railway は Railway-TC オブジェクトの、signal は任意の ID を持つ Signal オブジェクトの状態空間を取り出すように定義される。

演算 tcexpand は、信号機の防護範囲を示す軌道回路 ID のリストを元に、各軌道回路の状態を取得し、それらの論理和を得るための項を生成するための演算である。射影演算 signal では、この結果を参照し、信号の現示を決めている。

4.5.4 複線自動閉そくにおける安全性の検証

複線自動閉そく Auto-Sys オブジェクト上における安全性を、閉そくの概念を利用して検証する。Auto-Sys オブジェクトにおいて、常に閉そくが保たれていれば、列車が追突事故を起こすことはないことが示せる。

Auto-Sys オブジェクトに含まれている Section オブジェクトは、Auto-Sys の操作演算 move-train に伴って、その状態は、変化しないものと、in-train 、out-train の操作演算が適用されるものの 3 つがある。これらの場合に分けて、閉そくが保たれているかどうかを検証する。

まず、証明のために、以下のモジュール AUTOMATIC-BLOCKSYSTEM-PROOF を定義する。

```
op rw : -> Auto-Sys
op tr  : -> TrainID
op s1 s2 s3 : -> Section
ops sc1 sc2 sc3 : -> SectionID

op is-safety _ : Section -> Bool
op is-safety [ _ ] _ : SectionID Auto-Sys -> Bool

-- safety of Auto-Sys.
var Sc : SectionID
var Rw : Auto-Sys
eq is-safety [ Sc ] = is-safety (section Sc Rw) .

-- safety of section.
var S : Section
ceq is-safety S = true              if exist? S == false .
ceq is-safety (in-train S) = true   if exist? S == false .
ceq is-safety (out-train S) = true  if exist? S == true .
```

定数 rw は、任意の Auto-Sys オブジェクトを意味している。同様に、定数 tr は任意の TrainID を意味している。s1, s2, s3 は現在は安全であるという仮定を満たしている Section オブジェクトであり、sc1, sc2, sc3 はそれらの ID である。

Section オブジェクトを引数に取る、述語 is-safety は、Section オブジェクトが安全である場合に true を返す。安全である場合とは、仮定から安全であると導かれる初期状態、列車が存在しない Section オブジェクトに列車が進入した状態、及び、列車が存在する Section オブジェクトから列車が進出した場合の 3 通りであり、それを条件付き等式によって与える。

SectionID と Auto-Sys オブジェクトを引数に取る述語 is-safety は、Auto-Sys に含まれている任意の SectionID を持つ Section オブジェクトの安全性を示すための述語で、安全ならば true を返す。

sc1: 列車が存在せず、状態が変化しない閉そく区間

以下に示すのオブジェクトの観測を決定する等式を仮定として与え、すべての Auto-Sys オブジェクトの状態 rw において、述語 is-safety が true であることを示す。

```
eq [hyp1-1] is-safety s1 = true .
eq [hyp1-2] exist? sc1 = false .
eq [hyp1-3] section sc1 rw = s1 .
eq [hyp1-4] watch (signal (r2s where?
                (train Tr (railway (railway Rw)))) Rw) = R .
red is-safety [ sc1 ] rw .
red is-safety [ sc1 ] move-train tr rw .
```

等式 [hyp1-4] を信号の各現示 (R, Y, G) の場合に分けて、リダクションを繰り返す。これによって、閉そく区間 sc1 が常に安全であることを示せた。以下の各場合においても同様のリダクションによって、安全であることを示すことができる。

sc2: 列車が存在しており、move-train によって列車が進出する区間

```
eq [hyp2-1] is-safety s2 = true .
eq [hyp2-2] exist? sc2 = true .
eq [hyp2-3] exist? sc3 = true .
eq [hyp2-4] section sc2 rw = s2 .
eq [hyp2-5] watch (signal (r2s where?
                (train Tr (railway (railway Rw)))) Rw) = R .
red is-safety [ sc2 ] rw .
red is-safety [ sc2 ] move-train tr rw .
```

sc3: 列車が存在せず、move-train によって列車が進入する区間

```
eq [hyp3-1] is-safety s3 = true .
eq [hyp3-2] exist? sc3 = true .
eq [hyp3-3] section sc3 rw = s3 .
eq [hyp3-4] watch (signal (r2s where?
                (train Tr (railway (railway Rw)))) Rw) = R .
red is-safety [ sc3 ] rw .
red is-safety [ sc3 ] move-train tr rw .
```

第 5章

単線区間における自動閉そく

5.1　正面衝突の防止

単線区間では、列車が一本のレールの上を双方向に走ることになる。この場合、鉄道で最も忌み嫌われる正面衝突を起こす怖れがある (図 5.1) 。列車 A と列車 B は何もしなければ衝突するし、何らかの方法で衝突を回避したとしても、お互いの列車が邪魔をしてそれ以上先には進めない。

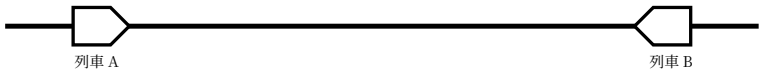

図 5.1: 正面衝突が起こる場合

そこで、列車交換の概念を導入する。列車交換に必要な最小限の設備を図 5.2 に示す。

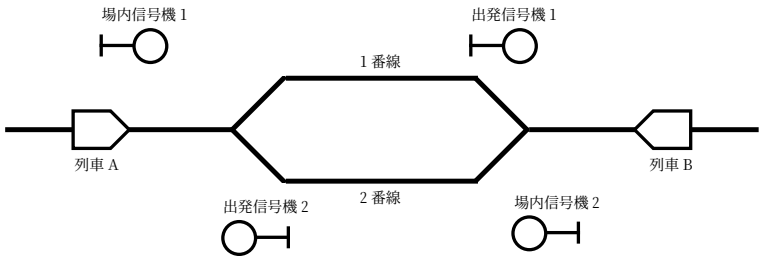

図 5.2: 最小限の行き違い設備

列車交換をするには、あらかじめ行き違いをする場所を決めておき、その入り口までは対向列車の動きとは無関係に列車を進める。交換駅手前に列車が到着すると、場内信号機に従って、列車は駅へ進入する。場内信号機は駅へ進入してよいかどうかを現示する信号機であり、分岐器が正しい方向に開通し、場内進路が構成されている時、進行信号が現示される。ただし、列車交換のような場合では、場内信号機はその次に位置する出発信号機が停止信号を現示しているので、その予告として注意信号が現示されることになる。図 5.2 では、場内信号機 1 は列車 A 、場内信号機 2 は列車 B が存在する方向から接近する列車に対する信号である。

出発信号機は、列車が駅から進出してよいかどうかを現示する信号機である。列車 A, B ともに交換駅に到着し、次の交換駅の入り口までの閉そくが完了すると、それぞれの出発信号機が進行信号となる。

図 5.2 の最小限の行き違い設備では、次のような運用が行われる。ここでは、列車 A が先に交換駅に接近した場合を例にとる。

1. 列車 A 側の分岐器を 1 番線側に開通させ、場内進路を構築し、場内信号機 1 を進行信号とする。

2. 1 と同時に場内信号機 2 を停止信号とし、列車 B の進入を抑止する。

3. 列車 A は進行信号に従い、1 番線に入り、停止する。

4. 列車 A の停止を確認した後、場内信号機 2 を進行信号とし、列車 B を 2 番線に入線させる。

この手順は、列車 B は駅手前で停止信号により待たされるかもしれないことを意味している。2. の手順を変え、場内信号機 2 にも進行信号を現示させると列車 B は待たされなくて済む。理想的には、列車 A と列車 B が同時に交換駅に進入すると、到着後も対向列車を待つことなく発車できるので、良いダイヤを作成することができる。

しかし、これには問題がある。列車は必ず止まれるとは限らないことを考慮しなければならない。例えば、列車 B が列車 A にわずかに遅れて交換駅に接近しつつある時、何かの拍子に列車 A が出発信号機までに停止できず、分岐器に差し掛かった列車 B に衝突するかもしれない。このような事故は未然に防止しなければならない。そこで、前述の手順の他に、安全側線、過走余裕、場内進入速度の低下などの解決策がある。

安全側線とは、過走した列車を砂利盛りなどに乗り上げさせ、列車を脱線させるための設備である。安全測線は出発信号機手前に設け、分岐器を普段は安全側線側にしておき、

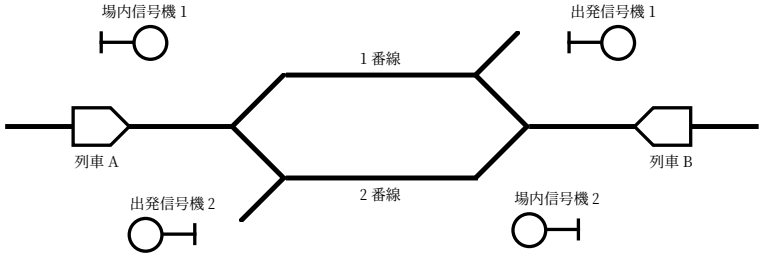

図 5.3: 安全側線

万一列車が過走した場合には、安全側線に進入させ、脱線させる。対向列車の到着が確認されると、分岐器を本線側に向け、出発進路を構成するというものである。

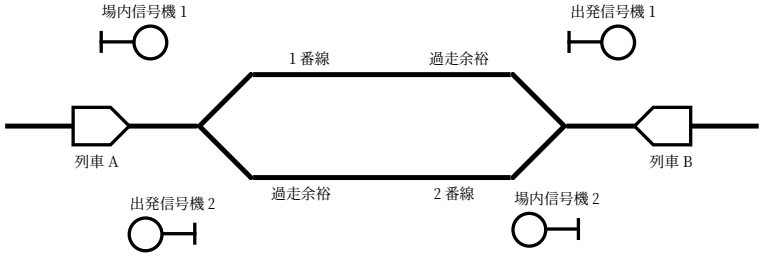

図 5.4: 過走余裕

過走余裕は、その名の通り、多少の過走では衝突しないように、十分な余裕距離を設けたものである。逆に、地形等の関係で過走余裕が確保できない場内には、進入速度を低下させるという方法もある、場内信号機に警戒信号を現示し、場内の進入速度を低下させ、過走そのものを防止するという方法もある。

5.2 単線自動閉そくの設備

単線区間における自動閉そくは、複線のものと同様の設備であるが、正面衝突を防止するため、対向列車を抑止するための設備が必要である。停車場間に一対の「方向てこ」を設け、これを操作することによって列車の運転方向を決定する。

図 5.5 の例では、甲 - 乙駅間の列車運転方向は左側 (乙駅から甲駅の方向) に設定されており、列車 B が停車場間を進行中である。両駅の方向てこの鎖錠表示灯が点灯しており、方向てこを操作しても、運転方向が変化することはない。

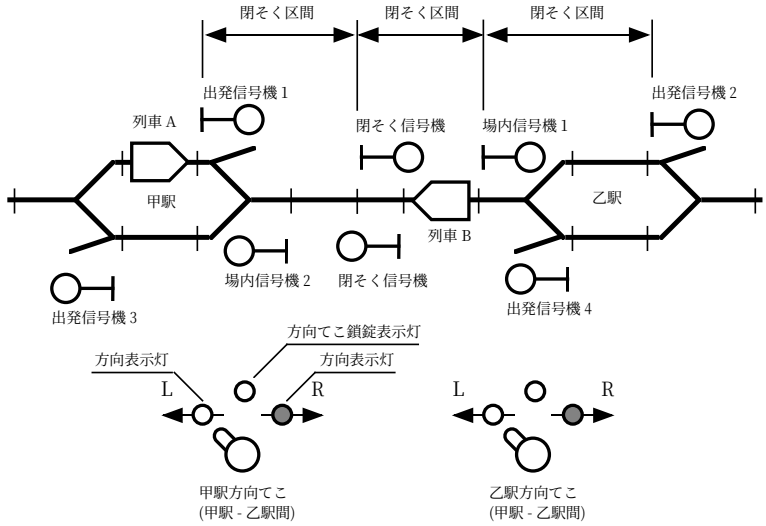

図 5.5: 単線区間における自動閉そくの設備

列車 B が甲駅に到着したので、今度は列車 A を甲駅から乙駅に運転したい。そこで、まずは、乙駅側が出発信号てこを定位 (停止信号) に戻す。すると、乙駅の出発信号機に停止信号が現示されるとともに、乙駅の方向てこは解錠される。 (図 5.6) 。

図 5.6: 乙駅の進行信号てこを定位に戻した時の方向てこの状態

次に乙駅の方向てこを受け (R) 側に転換する。甲駅側の方向てこも解錠される (図 5.7) 。

甲駅では、乙駅の操作を受けて、方向てこを出発 (R) に転換する。乙駅側の方向てこが鎖錠され、鎖錠表示灯が点灯する。 (図 5.8) 。

甲駅で列車 A を発車するために出発信号てこを反位に転換する。これによって甲駅の方向てこの鎖錠表示灯が点灯する (図 5.9) 。

方向てこによる鎖錠を唯一解除する方法は、送り側の駅の出発信号てこを定位に戻すことである。列車が受け側の駅に到着した場合、列車の突然の故障などで運転方向を変更したい時などには、この操作を行うことになる。いずれの場合であっても停車場間に列車が

図 5.7: 乙駅の方向てこを R に転換した時の方向てこの状態

図 5.8: 甲駅の方向てこを R に転換した時の方向てこの状態

図 5.9: 乙駅の出発信号てこを反位にした時の方向てこの状態

存在する時に解錠されてしまっては危険であるので、それをできないようにする仕組みが必要である。

5.3 方向てことその周辺の仕様

5.3.1 方向てこの仕様

前述のように、単線自動区間の駅間における列車の運転方向は、方向てこによって決定される。そこで、方向てこが取り得る値 Direction と、方向てこ自身を示すオブジェクト DirLever を定義する。

```
[ Direction ]
ops Left Right : -> Direction
```

ここで、値 Left は、方向てこが左を向いていることを意味し、Right は同様に右を意味している。次にオブジェクト DirLever のシグネチャを示す。

```
*[ DirLever ]*
bop which? _      : DirLever -> Direction
bop turn-left _   : DirLever -> DirLever
bop turn-right _  : DirLever -> DirLever
```

オブジェクト DirLever は、てこを左または右に転換するための操作演算 turn-left, turn-right と、てこがどちらを向いているかを知ることができる属性演算 which? を持つ。

5.3.2 出発信号てこの仕様

方向てこは出発信号てこと連査を持つので、出発信号てこも同様に定式化する。出発信号てこの仕様は、それが取り得る値 SigValue と、オブジェクト ExitLever によって与える。

```
[ SigValue ]
ops Go Stop : -> SigValue
```

ここで、値 Go は、出発信号てこが反位であり、駅から出発することができるという意味を持つ。 Stop は同様にてこが定位であり、停止することができないことを意味している。ただし、出発信号てこは、軌道回路との連査も持っており、出発信号てこを進行にしただけでは、出発信号機に進行信号が現示されるわけではない。これについては後で詳しく述べる。

```
*[ ExitLever ]*
bop which? _       : ExitLever -> SigValue
bop turn-go _      : ExitLever -> ExitLever
bop turn-stop _    : ExitLever -> ExitLever
```

オブジェクト ExitLever は、てこを進行または停止に転換するための操作演算 turn-go, turn-stop と、てこがどちらを向いているかを知ることができる属性演算 which? を持つ。

5.3.3 鎖錠表示灯の仕様

方向てこが鎖錠されると、鎖錠表示灯が点灯する。鎖錠表示灯の仕様は、それが取り得る値 LampValue と、オブジェクト Lamp によって与える。

```
[ LampValue ]
ops On Off : -> LampValue
```

ここで、値 On は、鎖錠表示灯が点灯していることを意味し、Off は同様に滅灯を意味している。

```
*[ Lamp ]*
bop light? _   : Lamp -> Lamp
bop lamp-on _  : Lamp -> Lamp
bop lamp-off _ : Lamp -> Lamp
```

オブジェクト Lamp は、init-lamp によって生成され、鎖錠表示灯を点灯または滅灯させるための操作演算 lamp-on, lamp-off と、鎖錠表示灯が点いているのか消えているのかを知ることができる属性演算 light? を持つ。

5.3.4 駅の仕様

方向てこは、出発信号てこ、鎖錠表示灯とともに駅に設置されている。そこで、DirLever, ExitLever, Lamp の各オブジェクトを合成した Station オブジェクトを定義する。

```
*[ Station ]*
bop lock _       : Station -> Station
bop unlock _     : Station -> Station
bop turn-left _  : Station -> Station
bop turn-right _ : Station -> Station
```

```
bop turn-go _    : Station -> Station
bop turn-stop _  : Station -> Station
--
bop lock-lamp    : Station -> Lamp
bop dir-lever    : Station -> DirLever
bop exit-lever   : Station -> ExitLever
```

lock, unlock は駅の方向てこを鎖錠・解錠するという意味の操作演算であり、Lamp オブジェクトの操作演算 lamp-on, lamp-off に対応している。turn-left, turn-right, turn-go, turn-stop は、オブジェクト DirLever, ExitLever の操作演算にそれぞれ対応している。lock-lamp, dir-lever, exit-lever は射影演算であり、合成前の各オブジェクトに分解する機能を持つ。

5.3.5 二駅間における鎖錠関係の仕様

方向てこは二駅間における列車の運転方向を決めるので、方向てこは隣接する駅との間で鎖錠関係を持つ。そこで、二つの Station オブジェクトから成る、Station-Pair オブジェクトを以下に示す。

```
*[ Station-Pair ]*
bop A-turn-left _  : Station-Pair -> Station-Pair
bop A-turn-right _ : Station-Pair -> Station-Pair
bop A-turn-go _    : Station-Pair -> Station-Pair
bop A-turn-stop _  : Station-Pair -> Station-Pair
--
bop B-turn-left _  : Station-Pair -> Station-Pair
bop B-turn-right _ : Station-Pair -> Station-Pair
bop B-turn-go _    : Station-Pair -> Station-Pair
bop B-turn-stop _  : Station-Pair -> Station-Pair
--
bop station-A      : Station-Pair -> Station-Pair
bop station-B      : Station-Pair -> Station-Pair
```

Station-Pair オブジェクトでは、二つの駅を A 駅、B 駅とし、これらの駅のうち、左側にあるのが A 駅、右側にあるのが B 駅という関係を仮定する。このオブジェクトにおいて、A 駅で許される操作演算は、A-turn-left, A-turn-right, A-turn-go, A-turn-stop の 4 つであり、B 駅も同様である。両駅の間の鎖錠関係は射影演算によって与えられる。以下に、射影演算に関する等式の一部を示す。

表 5.1: valid な状態

	A 駅			B 駅		
	方向てこ	出発信号てこ	鎖錠表示灯	方向てこ	出発信号てこ	鎖錠表示灯
状態 1	右	反位	点灯	右	定位	点灯
状態 2	右	定位	消灯	右	定位	点灯
状態 3	左	定位	消灯	右	定位	消灯
状態 4	左	定位	点灯	左	定位	消灯
状態 5	左	定位	点灯	左	反位	点灯

```
ceq station-A (A-turn-left SS) = turn-left (station-A SS)
    if  which? (dir-lever (station-A SS)) =/= Left
    and light? (lock-lamp (station-A SS)) =/= On .
ceq station-B (A-turn-left SS) = unlock (station-B SS)
    if  which? (dir-lever (station-A SS)) =/= Left
    and light? (lock-lamp (station-A SS)) =/= On .
```

これは A 駅の方向てこを左に転換した時の射影演算である。A 駅の方向てこが既に左に転換された後でなく、かつ A 駅の方向てこが鎖錠されていなければ、A 駅では方向てこが左に転換され、B 駅ではそれと同時に解錠されることを表現している。

5.3.6 方向てこの仕様の検証

定義した方向てこの仕様が、要求事項を満たしているかどうかを検証する。方向てこに関する要求事項は以下の二点である。

1. 隣接する二駅間において、対向列車を抑えることができる。言い換えると、出発信号てこを反位にすることができるのは、どちらか一方の駅である。

2. 出発信号てこを反位とした時は、方向てこを動かすことができない。

これを検証するために、方向てこ、出発信号てこ、鎖錠表示灯の観測の組を、このシステムの状態とする。この状態のうち、表 5.1 に示すものを valid な状態と定義とする。

表 5.1 で示す状態のうち、状態 1 ではA駅から出発可であり、状態 5 ではB駅から出発可である。状態 2 〜 4 では、出発信号機は停止信号であり、どちらの駅からも出発す

ることはできない。これらの状態それぞれにおいて、可能な操作を行って各オブジェクトの状態が遷移した時に、表 5.1 の状態の何れかに留まっているとすれば、両方の駅から同時に出発可となることはなく、要求事項 1 を満たしていることがわかる。

次に、Station-Pair 上の振舞等価関係 R を定義する。

```
-- Behavioural Equivalence on Station-Pair .
op _ R _ : Station-Pair Station-Pair -> Bool .
var S1 S2 : Station-Pair
eq S1 R S2 = dirlever  (station-A S1) =*= dirlever  (station-A S2)
         and exitlever (station-A S1) =*= exitlever (station-A S2)
         and locklamp  (station-A S1) =*= locklamp  (station-A S2)
         and dirlever  (station-B S1) =*= dirlever  (station-B S2)
         and exitlever (station-B S1) =*= exitlever (station-B S2)
         and locklamp  (station-B S1) =*= locklamp  (station-B S2) .
```

表 5.1 で示す状態を st1 〜 st5 として表現する。

```
ops st1 st2 st3 st4 st5 : -> Station-Pair .
-- Case: st1.
eq light? (lock-lamp (station-A st1)) = On .
eq light? (lock-lamp (station-B st1)) = On .
eq which? (dir-lever (station-A st1)) = Right .
eq which? (dir-lever (station-B st1)) = Right .
eq which? (exit-lever (station-A st1)) = Go .
eq which? (exit-lever (station-B st1)) = Stop .
-- Case: st2.
...
```

任意の S : Station-Pair が valid な状態にあることを判定する述語 isin-vaild-state を定義する。状態 S が st1 〜 st5 のどれかと観測等価ならば、状態 S は valid な状態のいずれかにある。

```
-- State S is in valid states.
op isin-valid-state _ : Station-Pair -> Bool .
var S : Stations .
eq isin-valid-state S = (S R st1) or (S R st2) or (S R st3) or
                        (S R st4) or (S R st5) .
```

任意の S : Station-Pair に対し、すべての操作演算を適用し、操作演算を適用した後も valid な状態にあることを示す述語 forall-action を定義する。

```
op forall-action _ : Station-Pair -> Bool .
eq forall-action S = isin-valid-state (A-turn-left S)
                and = isin-valid-state (A-turn-right S)
                and = isin-valid-state (A-turn-go S)
                and = isin-valid-state (A-turn-stop S)
                and = isin-valid-state (B-turn-left S)
                and = isin-valid-state (B-turn-right S)
                and = isin-valid-state (B-turn-go S)
                and = isin-valid-state (B-turn-stop S) .
```

述語 forall-action を用い、st1 〜 st5 のそれぞれの状態の Station-Pair に対して、すべての操作演算を適用した後も valid な状態にあることを、以下の項を簡約することによって示す。

```
reduce (forall-action st1) and (forall-action st2)
   and (forall-action st3) and (forall-action st4)
   and (forall-action st5) .

--> Should be true.
```

任意の操作演算を適用した後、状態 st1 〜 st5 にある Station-Pair は、仮定から valid な状態である。その状態にすべての操作演算を適用した後の、各 Station-Pair も状態 st1 〜 st5 のうちいずれかと等価な状態を持つので、Stations-Pair は valid な状態以外の状態に遷移することはない。

次に、鎖錠灯が点灯している状態 st1, st2, st4, st5 の各状態について、方向てこや出発信号てこが正しく鎖錠されているかどうかを検証する。

ここでは例として、st2 の場合のみ示す。st2 は B 駅の方向てこ鎖錠表示灯が点灯している。従って、B 駅の方向てこに対する操作演算 B-turn-left/right 、出発信号てこに対する操作演算 B-turn-go/stop の適用が、Station-Pair の状態を変化させないことを示す。

```
-- Case st2.
eq light? (lock-lamp (station-A st2)) = Off .
eq light? (lock-lamp (station-B st2)) = On .
eq which? (dir-lever (station-A st2)) = Right .
eq which? (dir-lever (station-B st2)) = Right .
eq which? (exit-lever (station-A st2)) = Stop .
eq which? (exit-lever (station-B st2)) = Stop .
```

```
reduce st2 R (B-turn-left st2) .
reduce st2 R (B-turn-right st2) .
reduce st2 R (B-turn-go st2) .
reduce st2 R (B-turn-stop st2) .

--> Should be true.
```

例えば、st2 と st2 に操作演算 B-turn-left を適用した後の Station-Pair が観測等価ならば、st2 において、操作演算 B-turn-left は Station-Pair の状態を変化させないことを示している。

このように、この仕様は、いかなる場合においても Valid な状態以外の状態に遷移することはない。また、両駅の方向てこが鎖錠されている状態 1 や状態 5 や、A 駅だけが鎖錠されている状態 4 、B 駅だけが鎖錠されている状態 2 において、鎖錠されている方向てこを操作することができないことを示した。以上のことから、方向てこの仕様は要求事項 1 および 2 を満たしていることがわかった。

5.4 単線自動閉そく (特殊)

5.4.1 信号との連鎖

方向てこによって運転する方向を決め、方向てこを鎖錠した後、すぐに列車を出発させてよいわけではない。軌道回路によって進路の開通を確かめた後でなければ、出発信号機を進行信号にすることはできない。そこで、単線自動閉そく (特殊) が用いられる。単線自動閉そく (特殊) は、自動 B と略される (以下では自動 B とする)。自動 B の配線略図を図 5.10 に示す。自動 B は、同一方向に続けて二箇列車を運転することのない線区に用いられる。

図 5.10 中で、線路上に示されている 3LT, 2RT, 11T, AT, 12T, 2LT, 3RT は軌道回路名である。A 駅からは 2RT 上に存在する列車が出発し、11T, AT, 12T, 3RT と進行する。B 駅からは 2LT 上に存在する列車が出発し、12T, AT, 11T, 3LT と動くとする。分岐器や駅の構造 (何番線まであるかなど) については、ここでは規定しない。

出発信号機 2R に進行信号が現示できる条件は、以下の通りである。

1. A 駅出発信号てこが反位
2. 軌道回路 11T, AT 上に列車がいない

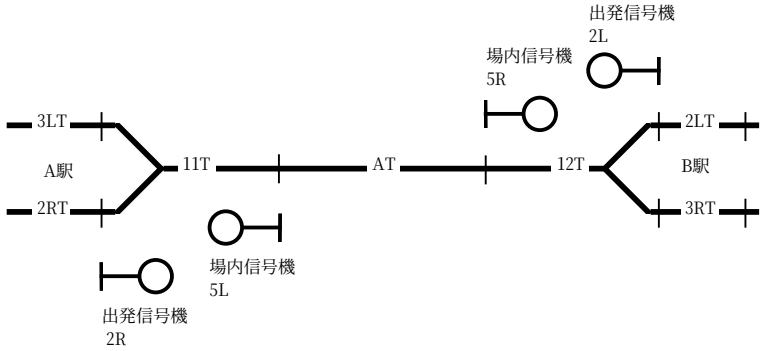

図 5.10: 典型的な単線自動閉そくの配線略図

では、2RT 上の列車が出発信号機 2R の進行現示に従って発車したとしよう。11T 上に列車が進むと、進行信号が現示できる条件 2 が満足されず、2R は停止現示となる。AT 上に進んだ時も同様である。これによって、後続の列車が 11T, AT に進入することを抑止する。これを閉路鎖錠と呼ぶ。

次に、A 駅から出発した列車が 11T あるいは AT 上にいる場合を考えてみる。A 駅の出発信号てこを誤って定位に戻してしまい、A 駅の方向てこが解錠されるようでは困る。なぜなら、A 駅の方向てこの転換が可能になり、それは B 駅の方向てこと出発信号てこが転換可能になることを意味する。これでは、列車が 11T 上を走行中に、B 駅の出発信号機 2L に進行が現示されてしまう怖れがある。AT にいる場合は仮に解錠され、B 駅の出発信号てこを反位にしたとしても、AT は出発信号機 2L の防護範囲に入っているため、進行信号が現示されることはないため、一見大丈夫のように見える。しかし、AT は A 駅を発車した列車の進路として設定されており、その列車が走行中に、その進路を取り上げたり、他の列車の進路と重なるなど、システム上で矛盾するような行為を行ってはならない。

以上のことから、列車が A 駅から出発し、11T, AT 上を走行中は、A 駅の出発信号てこを鎖錠する必要があることがわかる。また、出発信号機は進行と停止の二現示で済むこともわかる。このような鎖錠を進路鎖錠と呼ぶ。

一方、場内信号機に関しては、また違った注意が必要である。図 5.10 のシステムでは、場内信号機 5L の防護範囲は 11T と 3LT である。その先は車止めか、出発信号機が存在する。5L には、その先が車止めか、もしくは出発信号機が停止の場合は注意信号を、通過列車を運転する場合などに、出発信号機が進行の場合は進行信号を現示する必要があるので、三現示の信号機にしなければならない。

5.4.2 駅間コンポーネントの仕様

最初に、自動 B の A 駅から B 駅までの間をコンポーネントとし、モデル化する。このコンポーネントには 2RT, 11T, AT, 12T, 2LT の軌道回路が敷設されたレールオブジェクトを含んでいる。

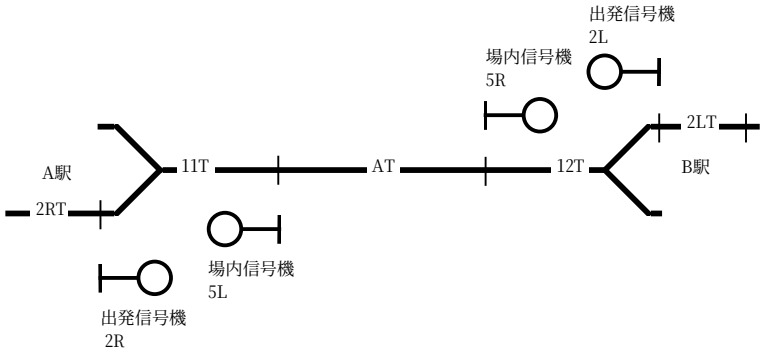

図 5.11: 自動 B の駅間コンポーネント

複線自動閉そくの仕様と同じ Railway-TC を利用し、それに出発信号機と場内信号機および方向てこを取り付ける。単線自動閉そくの AutoB のシグネチャを以下に示す。

```
*[ AutoB ]*
bop  move-train _ _    : TrainID AutoB -> AutoB
bop  A-reach-train _ _ : TrainID AutoB -> AutoB
bop  A-leave-train _ _ : TrainID AutoB -> AutoB
bop  B-reach-train _ _ : TrainID AutoB -> AutoB
bop  B-leave-train _ _ : TrainID AutoB -> AutoB
--
bop  A-turn-left _     : AutoB -> AutoB
bop  A-turn-right _    : AutoB -> AutoB
bop  A-turn-go _       : AutoB -> AutoB
bop  A-turn-stop _     : AutoB -> AutoB
bop  B-turn-left _     : AutoB -> AutoB
bop  B-turn-right _    : AutoB -> AutoB
bop  B-turn-go _       : AutoB -> AutoB
bop  B-turn-stop _     : AutoB -> AutoB
--
bop  A-exit-signal _   : AutoB -> Signal2
bop  B-exit-signal _   : AutoB -> Signal2
```

```
bop  A-entry-signal _  : AutoB -> Signal3
bop  B-entry-signal _  : AutoB -> Signal3
bop  stations _        : AutoB -> Stations
bop  railway-tc _      : AutoB -> Railway-TC
```

操作演算 move-train は、任意の TrainID を持つ列車を信号に従って動かす演算である。A-reach-train はレール 2RT に列車が到着した時に適用される操作演算であり、これによって Train オブジェクトが作られる。B-leave-train は 12T から B 駅へ列車を進めるため、この駅間コンポーネントから列車オブジェクトを削除するための操作演算である。B-reach-train, A-leave-train も同様の演算である。

出発信号機への射影演算 A-exit-signal や B-exit-signal 、あるいは出発信号てこを含む Stations に対する射影演算 stations を定義する際には、前述のような鎖錠関係を定義しなければならない。

5.4.3 駅間コンポーネントにおける安全性の検証

方向てこの仕様により、出発信号てこは A, B 両駅同時に反位とすることはできないことを既に示した。出発信号てこが反位である時に、出発信号機は進行現示となるため、列車が信号機に従っている限り正面衝突が起こらないことは自明である。

ここでは、軌道回路の状態と出発信号てこの間の鎖錠関係について、検証を行う。図 5.10 において、列車が A 駅から B 駅へ進行する場合を考える。2RT に停車していた列車は出発信号機 2R が進行になると発車することができるが、この状態では A 駅の出発信号てこを定位に戻すことができる。これは、例えば 2RT に停車中の列車に故障が発生したなどにより、ひとまず運転を取りやめ、B 駅で交換待ちをしている列車を A 駅が受け入れることを可能にするためである。

この状態を rw1 とすると、列車が 2RT に留まっている限り、常に A 駅の出発信号てこを定位に戻すことができることを余帰納法によって示す。

```
op rw1 : -> AutoB-Sys
eq where? (train tr (railway (railway-tc rw1))) = 2RT .
eq watch (A-exit-signal rw1))) = G .
-- 軌道回路の状態を仮定
eq exist? (trackcircuit 2RT (railway-tc rw2)) = true .
-- 方向てこの状態を仮定
eq which? (dir-lever (station-A (stations rw1))) = Right .
eq which? (dir-lever (station-B (stations rw1))) = Right .
```

```
-- 出発信号てこの状態を仮定
eq which? (exit-lever (station-A (stations rw1))) = Go .
eq which? (exit-lever (station-B (stations rw1))) = Stop .
-- 鎖錠灯の状態を仮定
eq light? (lock-lamp (station-A (stations rw1))) = On .
eq light? (lock-lamp (station-B (stations rw1))) = On .
-- A 駅の出発信号てこを反位にしてみる。
red which? (exit-lever (station-A (stations (turn-A-stop rw1)))) .
--> Should be Stop.
```

さて、列車が動き出し、11T (または AT) に入ったとする。この時、11T は出発信号機 2R の防護範囲であり、その中に列車がいるため、出発信号機 2R は停止信号となり、後続列車の進入を防止する。この状態では、出発信号てこを鎖錠し、方向てこを解錠できないようにする必要がある。この状態を rw2 とし、出発信号てこを定位に戻すことができないことを余帰納法によって示す (11T の場合。AT も同様) 。

```
op rw2 : -> AutoB-Sys
eq where? (train tr (railway (railway-tc rw2))) = 11T .
eq watch (A-exit-signal rw2))) = G .
-- 軌道回路の状態を仮定
eq exist? (trackcircuit 2RT (railway-tc rw2)) = true .
-- 方向てこの状態を仮定
eq which? (dir-lever (station-A (stations rw2))) = Right .
eq which? (dir-lever (station-B (stations rw2))) = Right .
-- 出発信号てこの状態を仮定
eq which? (exit-lever (station-A (stations rw2))) = Go .
eq which? (exit-lever (station-B (stations rw2))) = Stop .
-- 鎖錠灯の状態を仮定
eq light? (lock-lamp (station-A (stations rw2))) = On .
eq light? (lock-lamp (station-B (stations rw2))) = On .
-- A 駅の出発信号てこを反位にしてみる。
red which? (exit-lever (station-A (stations (turn-A-stop rw2)))) .
--> Should be Go.
```

列車が 11T と AT を通過し、12T (または 3RT) に入ったとする。この時、列車が既には出発信号機 2R の防護範囲で出ており、かつ場内信号機 5R に従って 3RT まで進むことができる。この状態では、出発信号てこが解錠される。この状態を rw3 とし、出発信号てこを定位に戻すことができることを余帰納法によって示す (12T の場合。3RT の場合も同様) 。

```
op rw3 : -> AutoB-Sys
eq where? (train tr (railway (railway-tc rw3))) = 12T .
eq watch (A-exit-signal rw3))) = G .
-- 軌道回路の状態を仮定
eq exist? (trackcircuit 12T (railway-tc rw3)) = true .
-- 方向てこの状態を仮定
eq which? (dir-lever (station-A (stations rw3))) = Right .
eq which? (dir-lever (station-B (stations rw3))) = Right .
-- 出発信号てこの状態を仮定
eq which? (exit-lever (station-A (stations rw3))) = Go .
eq which? (exit-lever (station-B (stations rw3))) = Stop .
-- 鎖錠灯の状態を仮定
eq light? (lock-lamp (station-A (stations rw3))) = On .
eq light? (lock-lamp (station-B (stations rw3))) = On .
-- A 駅の出発信号てこを反位にしてみる。
red which? (exit-lever (station-A (stations (turn-A-stop rw2)))) .
--> Should be Go.
```

これらによって、必要な鎖錠が行われていることが検証できた。列車の進行方向が B 駅から A 駅に設定されている場合の検証も、同様にして行うことができる。

第 6章

結論

6.1 本研究のまとめ

6.1.1 仕様記述

本研究では、鉄道信号システムのスキームに対して形式手法の適用を試みた。仕様化に際して、列車やレール、軌道回路や信号機をオブジェクトとして捉えるため、隠蔽代数によるオブジェクト記述をモデルとして採用し、隠蔽代数が扱える唯一の代数仕様言語である CafeOBJ を用いて、鉄道信号システムの仕様を記述した。

本研究では、鉄道信号において重要な概念である「閉そく」と「連動」に関する仕様を中心に、鉄道信号を捉え、閉そくの具体的な実装として、我が国の複線区間において採用されている「自動閉そく」と、単線区間の自動閉そくの一つである「自動閉そく (特殊) 」を取り上げ、それらを CafeOBJ で記述した。その際には、連動によって規定されている鎖錠関係を一貫して射影演算によって記述した。

6.1.2 安全性の検証

CafeOBJ のもう一つの特徴である、項書換えによる実行可能性を用い、記述した仕様を、余帰納法によって検証し、複線区間における自動閉そくと、単線区間における自動閉そく (特殊) について、特定の線路配置や線路の長さなどに依存することなく、信号システムのスキームのレベルにおいて、安全性を示した。

また、単線区間において、人手が介在する方向てこ、出発信号てこに関して、仕様によって与えられた鎖錠関係によって、危険な操作が行えないことを、余帰納法によって示した。

このようなプロセスを通じ、鉄道信号システムのスキーム設計の段階における形式手法や代数仕様言語 CafeOBJ の有効性を示した。

6.2 関連研究

鉄道信号に形式仕様を適用した研究の多くは、駅の連動論理の検証である。駅構内には信号てこなどの他、多数の分岐器やその進路上に設けられた軌道回路、場内信号機や出発信号機、構内運転や列車連結のための信号機など、数多くの保安装置が複雑な鎖錠関係を持っている。

これらの関係は、連動図表として仕様が与えられているが、この安全性を形式手法によって保証しようという研究が、世界各国で行われている。連動論理には様々な数学的基盤を背景にした形式手法が適用され、多くはモデル検査法による検証が行われている。これらの研究成果は、小規模な駅の検証においては十分実用化されている。

Morley は、プロセス代数と高階述語論理によって、イギリスの鉄道をモデル化し、連動論理の検証を自動化する研究を行った [11] 。我が国においては、川村らによって、時相論理と有限状態機械によって記述された仕様を元に、モデル検査法によって検証を効率的に行う方法が提案されている [12] 。

Bjørner らは、中国の鉄道システムを形式仕様言語 RAISE を用いて記述している。彼らは仕様記述の対象は鉄道信号に留まらず、列車の運行スケジュールや座席予約システムなども含まれており、最終的には鉄道システム全体を形式仕様によって記述することを目標としている [13] 。

また、鉄道システムには、人間のミスを機械が補い、機械の故障を人間が補うなど、人間系と機械の相互の関わり合いが多く見られる。筆者らは、鉄道システムにおけるこのような特性について、形式仕様の適用を試みた研究も行っている [14] 。

6.3 今後の展望と課題

6.3.1 自動閉そくの仕様記述

本研究では、自動閉そくとして分類されている閉そく方式のうち、複線区間において仕様されている自動閉そくと、単線区間で使用されている自動閉そく (特殊) について仕様を与えた。

これ以外にも、単線区間における自動閉そく、特殊自動閉そくがあり、これらの仕様は

与えなかった。ここでは、これらの方式に関する詳細には触れないが、自動閉そく (特殊) との相違点は、駅間が単一の閉そく区間ではなく、複数の区間に区切られていたり、コスト削減のため軌道回路による列車の存在検知を代替の方法によって実現していることにある。

本研究において与えた仕様のうち、自動閉そくにおける共通点は Railway-TC によって記述されており、これを元にして、他の方式の仕様についても、同様に記述・検証することができると考えられる。

6.3.2 非自動の閉そく

本研究では、自動閉そくを取り上げたが、非自動の閉そくも未だに多くの路線において使用されている。このようなシステムも、形式的に記述し、安全性を検証する必要がある。

非自動の閉そくについては、一部区間に軌道回路が敷設されておらず、その区間の開通状況の確認は、駅員の黙視に頼っている。このようなシステムを記述するには Railway-TC の仕様を再利用することはできず、Bare-Railway を元にして、新たなシステムを与えなければならない。

しかし、安全性の検証に関しては、自動閉そくの場合と同様に、余帰納法によって行うことができると考えられる。非自動の閉そくにおいても、駅間の開通を保証する閉そく装置と信号機の間には鎖錠関係を設け、危険な操作を行えないようにする必要がある。しかし、実際には、一部の方式において、鎖錠関係が不十分であり、信号システム上の欠陥が指摘されている [9] 。このような仕様を記述することは、形式仕様の検証過程が欠陥を洗い出す能力を持つという実例になり、意義のあることと考えられる。

6.3.3 代用閉そく

通常使用している閉そくシステムが、工事の影響や故障などで使えない時にも、路線が不通になってしまわないように、鉄道には代用閉そくに関する規定が存在する。このような、普段使わないシステムについても、形式手法を適用し、いざ必要とされる時に備えておく必要がある。

6.3.4 新しい信号システムへの適用

今日、鉄道の高速化に伴って新たな信号システムが必要とされている。例えば、北越急行ほくほく線では、160km/h 運転に対応するため、前方の進路の開通状況や、160km/h 運転可能な車両が接近するという条件により、信号機の現示を上げ (現示アップと呼ばれる) 、高速進行信号が現示される仕組みが追加されている。

一方で、新幹線や山手線などの ATC (Automatic Train Control) 区間における車内信号のようなシステムもある。ATC では信号はその区間で出してよい最高速度として現示され、もし列車がそれよりも速度超過の状態にあるならば、ATC が自動的にブレーキをかけ、衝突を回避する。

このようなシステムにおいても、あらゆる場合において閉そくが確保され、危険な操作が行えないように鎖錠関係が定義されていることを検証するために、本研究で行った手法は有効であると考えられる。

6.3.5 新しい設計手法の確立

今日の鉄道は、新しい駅を設計する度に連動論理の設計・検証を行っている。このような設計手法はある程度方法論が確立しているが、小さな駅でも複雑な検証が必要であり、大きな駅では検証すべき進路数が膨大となり実用的ではないと言われている。

そこで、鉄道信号のスキームを記述し、いくつかの線路コンポーネントを定義し、その安全性を検証する。このような検証済みのコンポーネントを集めて、安全な鉄道システムを設計するという手法が考えられる。本研究の成果を元に、このような新しい設計方法の確立は、複雑で大規模な鉄道システムの実現を容易なものにすると期待される。

謝辞

本研究を終始御指導下さった二木厚吉先生に感謝いたします。また、議論にお付き合いいただき、有益な助言をして下さった渡部卓雄先生、緒方和博先生、森彰先生に感謝いたします。CafeOBJ の仕様記述に際し、様々な疑問に快く答えて頂いた飯田周作氏、松本充広氏に感謝いたします。また、公私ともどもお付き合いいただいた言語設計学講座の皆様にお礼を申し上げます。

参考文献

[1] Răzvan Diaconescu, Kokichi Futatsugi, CafeOBJ Report, World Scientific, 1998.

[2] Shusaku Iida, Michihiro Matsumoto, Răzvan Diaconescu, Kokichi Futatsugi, Dorel Lucanu, "Concurrent Object Composition in CafeOBJ", JAIST Research Report, IS-RR-98-0009S, 1998.

[3] Răzvan Diaconescu, Kokichi Futatsugi, "Logical Semantics for CafeOBJ", JAIST Research Report, IS-RR-96-0024S, 1996.

[4] Joseph Goguen and Grant Malcolm, A hidden agenda, Technical Report CS97-538, UCSD Technical Report, April 1997.

[5] Shusaku Iida, Kokichi Futatsugi, and Takuo Watanabe, Algebraic specification of distributed systems based on concurrent object-oriented modeling, In Elie Najm and Jean-Bernard Stefani, editors, Formal Methods for Open Object-based Distributed Systems, Chapman & Hall, 1996.

[6] R. Hennicker, Context induction: a proof principle for behavioural abstractions, In A. Miola, editor, Design and Implementation of Symbolic Computation Systems, International Symposium DISCO 1990, number 429 in LNCS. Springer-Verlag, 1990.

[7] 吉武勇, 明本昭義, 運転保安設備の解説, 第 6 版, 日本鉄道図書株式会社, 1984.

[8] 塩沢寛, 鉄道の運転ルールの解説, 第 6 版, 日本鉄道図書株式会社, 1997.

[9] 永瀬和彦, 鉄道事故はなぜ起きるのか, 鉄道ジャーナル, No.385, 1998, pp. 58-64.

[10] 白土義男, 信号システムの移り変わり, 鉄道ピクトリアル, 鉄道図書刊行会, Vol. 48, No. 7, 1998, pp. 19-27.

[11] Matthew John Morley, Safety Assurance in Interlocking Design, Doctor of Philosophy University of Edinburgh, 1996.

[12] 川村正, 金森直, 鉄道信号システムの連動論理検証, 情報処理学会第 57 回全国大会, 講演論文集 (1), 1998, pp. 240-241.

[13] Dines Bjørner, Chris W. George, Bo Stig Hansen, Hans Laustrup, Søren Prehn, A Railway System, Coodination '97 Case Study Workshop Example, UNU/IIST Report No.93, 1996.

[14] 清野貴博, 代数仕様言語 CafeOBJ における高信頼システムの記述に関する一考察, 情報処理学会第 57 回全国大会, 講演論文集 (1), 1998, pp. 238-239.