博 士 論 文

項書換えを用いた安全性検証の組織化

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

清野 貴博

2003 年 3 月 31 日

要 旨

ソフトウェア工学において，状態機械は重要な計算モデルの一つである．高信頼システムの設計では，作成するシステムを状態機械でモデル化し，それが望ましい性質を持つことを科学的に検証することが，信頼性確保に有用であるとされている．状態機械は，大きく有限状態機械と無限状態機械に分けられる．帰納的に定義される状態機械のモデルは，無限状態を持つ機械をモデル化でき，状態遷移の回数に関する帰納法によって，その性質を検証できる利点を持つ．反面，検証の完全な自動化が困難であり，検証が成功するかどうかは，検証者の知識や経験に依存するという欠点を持つ．

無限状態機械が持つ性質を帰納法によって半自動的に検証するツールは，数多く提案されている．代数仕様言語 CafeOBJ とその処理系は，それらの一つである．CafeOBJ 処理系は，CafeOBJ で記述された状態機械の仕様を項書換え系と見なして，簡約による等式推論を行い，様々な性質を検証できる．CafeOBJ 処理系を用いた検証では，簡約以外の作業は検証者が行わなければならず，その主たる作業は適切な場合分けをすることである．しかし，実際の検証においては，膨大な場合分けを要することが多く，こうした作業を人手で処理することは現実的ではない．このため，場合分けを処理系により支援することが，有効であると考えられる．

本論文では，CafeOBJ 処理系によって場合分けを支援する手法を提案する．本論文で取り扱う“場合”とは，状態空間を分割して作った状態の集合である．最初に，基礎となる手法として，場合を項の組み合わせで表現する手法を提案する．これによって，次の二つの手法を現在の CafeOBJ 処理系で実現できる．一つ目は，検証に用いる場合分けをマトリクス状に整理する手法である．これは場合分けの網羅性を向上させる．このマトリクスは，状態機械の仕様に由来する軸と，表明に由来する軸から成り，後者をパラメタとしている．このため，表明ごとにマトリクスを再利用できる．二つ目は，検証を進める上で有効な場合分けや補題を発見するための手法である．前述のマトリクスだけでうまく検証できないときに，機械的に場合を生成し検証を試みることで，検証を進める手掛かりとなる場合分けの候補を発見できる．検証者は，見つかった場合分けを，前述のマトリクスに追加することで，検証を進めることができる．

なお，これらの手法を，実際に使用されているいくつかの鉄道信号システムの安全性検証に適用し，本手法の有効性を確かめた．

目次

図目次

第 1 章

序論

1.1 背景

情報技術の進歩とともに，我々の社会は計算機システムへの依存度を高めている．特に，計算機システムの重要な構成要素であるソフトウェアの信頼性確保は，急務である．

形式手法 (*formal methods*) はソフトウェアの信頼性確保に効果的であるとされている．形式手法とは，数学的基盤を持つソフトウェアの設計手法である．形式手法によって記述されたソフトウェアの仕様を、形式仕様 (*formal specifications*) と呼び，形式仕様を記述するための言語を形式仕様言語 (*formal specification languages*) と呼ぶ．形式仕様は，自然語による記述に比較して，厳密性，無矛盾性，非曖昧性の点において優れている．形式仕様のこうした特徴は，実際にソフトウェアを作る前に，形式仕様に基づく議論によってソフトウェアが持つ性質を科学的に解析できるという利点を生む源泉である．

形式仕様言語は，大きく，モデル指向の言語と代数指向の言語に分けられる．モデル指向の言語では，Z[32]，VDM[40] が，代数指向の言語では，OBJ3[14] が有名である．代数指向の形式仕様言語は，自由にデータ型を定義でき，モデル指向の言語に比較して，より問題に適した抽象度を選んで仕様記述ができるという利点があるとされている．CafeOBJ[1][11][20] は，OBJ3 を祖とする代数指向の仕様記述言語である．CafeOBJ は，主に始代数 (*initial algebra*) と隠蔽代数 (*hidden algebra*) を基盤とし，それらを組み合わせて仕様を記述できる．CafeOBJ で記述された仕様は，項書換え系 (*term rewriting systems*)[4] とみなすことができ，簡約によって記述した仕様を記号実行できる．CafeOBJ には処理系 [1] が存在し，処理系がこれを行う．仕様が実行可能であるという特徴により，仕様がある性質を持つかどうかを検証する際に，処理系による半自動的な検証が行える．

現実の例題を取り上げ，その仕様を CafeOBJ で記述し，その性質を検証すること

で，CafeOBJ が現実の例題にも適切に応用できることを示す研究も盛んに行われている [29][30][31][33][34]．近年特に，並行システム (*parallel systems*) や分散システム (*distributed systems*) を例題とした研究が盛んである [29][30][31][33][34]．こうしたシステムが持つ性質を議論する際には，状態機械 (*state machines*) としてモデル化するアプローチが一般的である [22][23][21]．緒方らが提案する振舞遷移機械 (*Observational Transition Systems*)[29] は，このようなモデルの一つであり，CafeOBJ での記述が容易であるという特徴を有する．状態機械が有する性質のうち，特に重要であるとされるものに，安全性 (*safety properties*) と活性 (*liveness properties*) がある．[29][30][31][33][34] では，対象とするシステムを振舞遷移機械でモデル化し，それを CafeOBJ で記述し，そのシステムが安全性や活性を有するかどうかを CafeOBJ 処理系による簡約によって検証している．こうした実験によって，これらのアプローチが現実の問題に対しても適切に応用できることが示されている．

1.1.1 振舞遷移機械

ここでは，振舞遷移機械の定義について簡潔に述べる．より正確な定義は，2 章で与える．最初に，振舞遷移機械でモデル化しようとするシステムを十分表現できる状態空間 Υ の存在を仮定する．状態は，状態空間の元である．振舞遷移機械 $\mathcal{S}$ は，観測の集合 $\mathcal{O}$，初期条件 $\mathcal{I}$，遷移規則の集合 $\mathcal{T}$ の 3 つ組で与える．各観測は，状態を構成する (一部の) 情報を型付きデータとして取得することを表している．より具体的には，各観測 $o \in \mathcal{O}$ は，状態を取り，型付きデータを返す関数である．初期条件は，各観測の初期値を決める条件である．初期条件を満たす状態を，初期状態と呼ぶ．遷移規則は，振舞遷移機械の状態遷移を表現する．各遷移規則 $\tau \in \mathcal{T}$ は，状態を取り，次の状態を返す関数である．

振舞遷移機械の実行は，初期状態から始まり，実行の各時点において非決定に遷移規則 $\tau \in \mathcal{T}$ が一つ選択され，適用され続ける．このようにして得られた状態列 $u_0, u_1, \ldots$ を $\mathcal{S}$ の計算と呼ぶ．

本論文では，振舞遷移機械の安全性と，その定義で用いる到達可能性について議論し，活性については扱わない．これは，活性に比較して安全性がより基本的かつ重要な性質であるからである．$\mathcal{S}$ のある状態 $u \in \Upsilon$ が到達可能であるとは，状態 u が現れる $\mathcal{S}$ の計算が存在することである．ある $\mathcal{S}$ が安全性を有するとは，望まない状態に決して到達しないことである．つまり，安全性は到達可能な任意の状態 $u \in \Upsilon$ において，u が望まない状態でないことを表現する状態に関する述語 p が成立することを示せばよい．

1.1.2 安全性の検証

[29][30] では，安全性の検証に，遷移規則の回数に関する帰納法 (以下では，単に帰納法とする) を用いている．帰納法による安全性検証は，無限の状態を有する状態機械に適用可能である，項書換えなどの比較的少ない計算資源で実現できる推論機構を用いて検証できるという長所がある．反面，検証の完全自動化が困難であり，検証が成功するかどうかは検証者の技量や経験に依存するという短所を有する．安全性の検証手法には，他にモデル検査法 (*model checking*)[10] が有名である．モデル検査法は，検証が全自動で行われるため，検証者の技量や経験を無視できるという長所を有する．反面，状態数が有限である状態機械に対してだけ適用が可能である，膨大な計算資源が必要であるという短所を有する．振舞遷移機械は，無限の状態を有する状態機械を表現できるので，振舞遷移機械の安全性を検証する手法としては，帰納法が適している．

図 1.1は，帰納法による安全性検証を木によって概観したものである．根は安全性検証全体を表している．帰納法を用いた検証手続きは，大きく基底段階と帰納段階の 2 つに分けられる．これらの推論を根から深さが 1 の各節で表す．基底段階では，安全性 p が任意の初期状態 $u_0 \in \Upsilon$ において成立するかどうかを推論する．帰納段階では，各遷移規則 $\tau_1, \tau_2, \ldots \in \mathcal{T}$ について[*1]，次の推論を行う．p が成立する任意の状態 $u \in \Upsilon$ について，$\tau_1, \tau_2, \ldots \in \mathcal{T}$ で表現された各遷移規則を適用した後の状態において，p が成立するかどうかを推論する．以上のすべての推論において，p が成立することが帰結できたならば，安全性 p が成り立つ．

帰納法による検証では，場合分けを伴う議論を要することがある．場合分けが必要になる主な理由は，検証に必要な情報の不足である．場合分けに関する推論は，根から深さが 2 以上である各節で表す．例えば，τ_2 に関する帰納段階の推論において，状態に関するある述語 q に関する情報が不足しているとする．q が取りうる値は，真または偽の 2 値である．情報が不足しているとは，ここでは q の真偽が決定できないことである．そこで，図 1.1 では，q が真および偽を仮定することで，それぞれの場合に分けて議論している．この場合分けによって，Υ は 2 つの状態の集合に分割され，それぞれの場合について議論することで，Υ 全体について議論し尽くすことができる．これらの議論において，なお情報が不足するときには，さらに別の述語 q' について場合分けを行うといったことを繰り返す．このように，場合分けによって検証を進めることができる．

[*1] 振舞遷移機械は，無限の遷移規則を有するが，実際の検証では，これを有限個の遷移規則で表現するための手法を用いる．

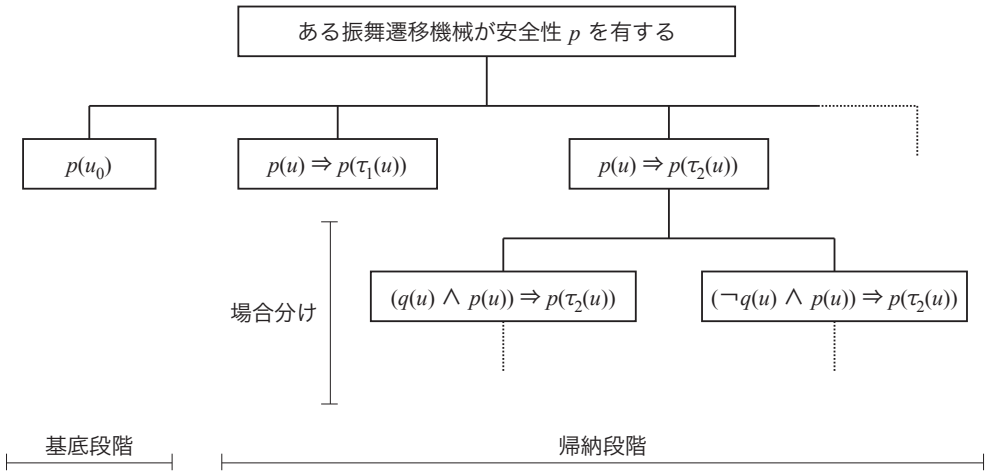


図 1.1: 帰納法による安全性検証

また，ある葉における推論結果が偽になることがある．これは、次に挙げるどちらを示している．一つ目は，その葉において議論している場合が反例であることを示している．もう一つは，もしくは，その場合が到達不能であることを示している．どちらであるかは，検証者の判断に委ねられるが，後者であると判断したならば，その場合が到達不能であることを示すことによって，検証を続けることができる．

CafeOBJ を用いて，帰納法を用いた検証を行うには，図 1.1で示した各葉について，証明譜 $(proof\ score)$ と呼ばれる小さなコードをそれぞれ記述する．これらの証明譜を CafeOBJ の処理系を用いて簡約することで，各葉が表している推論を行うことができる．これらの各簡約結果が，すべて所望の結果 (典型的には真を表す項) となるならば，その検証は成功したことがわかる．

1.2 研究の目的

前述したように，現実の例題を用いた事例研究によって，振舞遷移機械と CafeOBJ を用い，安全性検証を行うアプローチの有効性が示されている．一方で，これらの事例研究における検証 CafeOBJ の処理系が果たした役割は，検証者が記述した証明譜に基づき，簡約による推論を行ったのみである．つまり，場合分けなどの作業は人手で行っていた．

帰納法による検証では，人手の介在を完全に取り除くことはできないが，少ない方が望ましい．これは，検証に人手が介在することによって，考慮すべき場合を見落としたり，論理の飛躍を招くおそれがあるためである．これまでの CafeOBJ を用いた検証では，図 1.1で示した各葉について証明譜を記述していた．このため，検証者はどのような場合分けを用いて証明譜を記述したかを自分で管理し，すべての場合を尽くすように注意深く証

明譜を記述する必要があった．

本論文では，状態機械の特に重要な性質であると考えられる安全性に焦点を当てる．図1.1で示した検証において，CafeOBJ 処理系に場合分けを行わせることで，検証者を支援することを目的とする．

本論文では最初に，場合を項で表現する手法を提案する．これによって，次の二つの手法を現在の CafeOBJ 処理系で実現できる．

検証に必要な場合分けを一つのマトリクス状に整理する手法　このマトリクスは，状態機械の仕様に由来する場合分けを配した軸と，表明に由来する場合分けを配した軸から成る．マトリクスの各要素は，これらの軸から作られる場合を表し，すべての要素が表す場合について議論をし尽くすことによって，すべての場合について議論したことが保証できる．つまり場合分けの網羅性を向上させる．また，一般に一つの状態機械に対して検証したい表明は複数あるので，後者の軸をパラメタとし，マトリクスを再利用することができる．再利用したマトリクスが表現している場合分けでなお不足するときは，表明毎に固有の場合分けを簡単に追加することもできる．

検証を進める上で有効な場合分けや反例を発見するための手法　マトリクスで表現された場合分けで充分議論し尽くせることは少なく，より精密な場合分けが必要になる．このとき，機械的に場合分けを生成し検証を試みることで，検証を進める手掛かりとなる場合分けの候補や反例を発見できる．検証者は，見つかった場合分けを前述のマトリクスに追加したり，そこから補題を推測し，それが成立することを示すことによって，検証を進めることができる．

これらの手法についての評価は，現実の例題を取り上げ，この手法に基づき安全性を検証する実験を行うことで，それが適切に応用でき，検証者の支援が行えることを確認することによって行う．例題には，特に安全性が重要である高信頼システムの事例として，いくつかの鉄道信号システムを用いる．これらの鉄道信号は，現在，我が国の鉄道において，実際に使用されているものである．

1.3 本論文の構成

本論文は，以下の構成とする．

2 章　議論の準備として，振舞遷移機械の定義と，CafeOBJ の概要について触れ，CafeOBJ の構文や証明譜の記述法について，小規模な例題を元に簡潔に記述する．

3 章　振舞遷移機械を CafeOBJ で記述するための具体的手順を記述する．これまで，比較的その場限りの記述が用いられてきたが，本論文では振舞遷移機械を記述するための統一的な記法を提案する．

4 章　振舞遷移機械の安全性検証を組織化するための手法について述べる．最初に，場合を項で表現するための手法を提案する．これによって実現可能となる二つの手法もまた提案する．一つ目は，場合分けの網羅性を保証するために，検証に用いる場合分けをマトリクス状に整理する手法である．二つ目は，前述の手法でうまく検証できないときに，検証を進める上で有効な場合分けや反例を発見するための手法である．3 章と 4 章では，解説のための例題として簡単な鉄道信号システムと取り上げる．

5 章　事例研究の一つとして，複線区間の鉄道信号システムを取り上げ，本手法の適用し，仕様記述および安全性検証を試みる．

6 章　事例研究の一つとして，単線区間の鉄道信号システムを取り上げ，本手法の適用し，仕様記述および安全性検証を試みる．5 章と 6 章で行う実験を通して，本手法が現実の例題において，検証を支援する効果があることを確かめる．

7 章　本論文によって得られた知見について，関連研究と比較しながら，まとめる．

第 2 章

準備

本章では，最初に並行システムや分散システムの計算モデルとして用いる振舞遷移機械の定義について述べる．次に，振舞遷移機械の記述言語および検証支援系として，代数仕様言語 CafeOBJ の概略について触れる．

2.1 振舞遷移機械

緒方らが提案する振舞遷移機械 (*Observational Transition Systems*)[29][30] は，UNITY[8] を祖とし，並行システムや分散システムを状態機械としてモデル化できる．状態機械は，これらのシステムの抽象モデルとして広く用いられている [8][23][21] が，振舞遷移機械は，モデル化するシステムをブラックボックスと捉え，その外部から観察できる情報だけに基づいて，状態機械の性質について議論できる．このことは，内部の実装に依存しない，抽象度の高いモデル化を可能としている．

最初に，振舞遷移機械の状態と，各々の状態を元とする集合である状態空間について定義する．あるシステムをモデル化した振舞遷移機械 $\mathcal{S}$ について，$\mathcal{S}$ を適切に扱える十分な状態空間 Υ の存在を仮定する．振舞遷移機械 $\mathcal{S}$ の状態 u は，状態空間 Υ の元である．次に，振舞遷移機械をブラックボックスと見なし，その状態 (の一部分) を外側から調べることができる，いくつかの観測の集合 $\mathcal{O}$ を設ける．各観測 $o \in \mathcal{O}$ は，型付きの値を返す．

次に，振舞遷移機械の 2 つの状態 $u_1, u_2 \in \Upsilon$ の等価性を定義する．振舞遷移機械の状態間の一般的な等価性は，振舞等価性 [14] によって与えることができる．しかし，本論文で取り上げる事例では，より簡単な等価性だけで十分に議論できるため，状態の等価性は次のように定義する．

定義 2.1. 状態の等価性

2 つの状態 $u_1, u_2 \in \Upsilon$ の等価性は，各観測の値の組で識別できるとし，Υ 上の述語 $=_s$ で表現する:

$$u_1 =_s u_2 \text{ iff } \forall o \in \mathcal{O}.o(u_1) = o(u_2)$$

ただし，各観測の返値の型 D について，その元 $d_1, d_2 \in D$ の等価性を判定する D 上の述語 $=$ が適切に与えられていることを仮定する．

次に，振舞遷移機械の具体的な定義を述べる．

定義 2.2. 振舞遷移機械

振舞遷移機械 $\mathcal{S}$ は，観測の集合 $\mathcal{O}$，初期条件 $\mathcal{I}$，および遷移規則の集合 $\mathcal{T}$ の組 $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ で定義する:

観測 $\mathcal{O}$ 各 $o \in \mathcal{O}$ は，状態を引数に取り，その状態を構成するデータを返す関数 $o : \Upsilon \to D$ である．ただし，各 o の値域は，それぞれ異なってもかまわない．

初期条件 $\mathcal{I}$ $\mathcal{I}$ は，各 $o \in \mathcal{O}$ の初期値を決める．

遷移規則 $\mathcal{T}$ 各 $\tau \in \mathcal{T}$ は，ある状態を取り，次の状態を返す関数 $\tau : \Upsilon \to \Upsilon$ である．各 $\tau \in \mathcal{T}$ の意味は，効果と効力条件の組で定義する．効果は，τ による各 $o \in \mathcal{O}$ の変化である．効力条件は，述語 $c_\tau : \Upsilon \to \{\text{true}, \text{false}\}$ である．今，ある状態 u において，τ が適用されたとする．もし，$c_\tau(u)$ が真であるならば，各 $o \in \mathcal{O}$ の値は，τ の効果で定義した値に変化する．そうでないならば，各 $o \in \mathcal{O}$ の値は，変化しないと定義する．

例えば，温度を実数 R で返す観測 $thermo$ は，次のように定義できる:

$$thermo : \Upsilon \to R$$

例えば，観測 $thermo$ の初期値が 20.0 であることは，次のように定義できる:

$$\forall u \in \Upsilon.thermo(u) = 20.0$$

例えば，40.0 を上限として，温度を 0.1 上昇させる遷移規則 $raise : \Upsilon \to \Upsilon$ を考える．2 つの状態 $u, u' \in \Upsilon$ があり，これらに次の関係 $u' =_s raise(u)$ が成り立つとすると，効果と効力条件は，次のように定義できる:

効　果:$thermo(u') = thermo(u) + 0.1$

効力条件:$thermo(u) < 40.0$

類似した観測や遷移規則

観測や遷移規則のうち，類似したものを，添字を伴った $o_{i_1,\ldots,i_m}$ や $\tau_{j_1,\ldots,j_n}$ として定義することがある (ただし，$m,n \geq 0$)．データ D, D_k $(k = i_1,\ldots,i_m,j_1,\ldots,j_n)$ の存在を仮定すると，添字付きの演算は，観測の集合 $\{o_{i_1,\ldots,i_m} : \Upsilon \to D | i_1 \in D_{i_1},\ldots,i_m \in D_{i_m}\}$ や，遷移規則の集合 $\{\tau_{j_1,\ldots,j_n} : \Upsilon \to \Upsilon \mid j_1 \in D_{j_1},\ldots,j_n \in D_{j_n}\}$ を表している．

例えば，温度が $r \in R$ 変化することを表現する遷移規則を $\{raise_r : \Upsilon \to \Upsilon \mid r \in R\}$ のように定義する．$r < 0$ の場合も考慮し，温度の下限を 0.0 とした条件を加えて，効果と効力条件を次のように定義できる:

効　　果: $thermo(u') = thermo(u) + r$

効力条件: $thermo(u) + r \leq 40.0 \ \wedge \ thermo(u) + r \geq 0.0$

定義 2.3. 振舞遷移機械の計算

振舞遷移機械の実行は，初期状態から始まり，各実行のステップで遷移規則の 1 つが非決定的に選択され，実行される．この実行から，状態の無限列の集合が得られ，これを振舞遷移機械の計算と呼ぶ．より正確には，次の 2 つの条件[*1] を満たす状態の無限列 $u_0, u_1, \ldots$ の集合である:

- 開始性：状態 u_0 において，各 $o \in \mathcal{O}$ は $\mathcal{I}$ を満たす．
- 一貫性：すべての $i \in \{0,1,\ldots\}$ において，$u_{i+1} =_s \tau(u_i)$ となる遷移規則 $\tau \in \mathcal{T}$ が存在する．

振舞遷移機械の性質として重要なものは，安全性 (*safety properties*) と活性 (*liveness properties*) である．ここでは，特に安全性だけを取り上げる．これは，安全性が活性に比較してより基本的な性質であることと，本論文で取り上げた事例において，安全性が極めて重要な性質であるからである．振舞遷移機械 $\mathcal{S}$ の安全性とは，直感的には，$\mathcal{S}$ が望まない状態に決して陥らないことである．

振舞遷移機械 $\mathcal{S}$ の安全性，およびその定義で用いる到達可能性は，以下の定義とする．

定義 2.4. 到達可能性

状態 $u \in \Upsilon$ が現れる $\mathcal{S}$ の計算があるとき，u は到達可能である．

定義 2.5. 安全性

$\mathcal{S}$ のすべての到達可能な状態 $u \in \Upsilon$ において，述語 $p : \Upsilon \to \{\text{true}, \text{false}\}$ が真である時，

[*1] オリジナルの振舞遷移機械では，これらの 2 つの条件に加えて公平性 (*fairness*) の条件が加わるが，本論文では活性 (*liveness*) を扱わないので，定義から除いている．

$\mathcal{S}$ は安全性 p を有する．ただし，$p(u)$ は，状態 u のときの観測の値の組だけで決定可能な述語である．

本論文では，$\mathcal{S}$ が安全性 p を有することを，遷移規則の回数に関する帰納法を用いて検証する．帰納法の手続きは，以下の通りである:

- 基底: $\mathcal{I}$ を満たす任意の状態 $u \in \Upsilon$ で，$p(u)$ が成立することを示す．
- 帰納段階: 任意の到達可能な状態 $u \in \Upsilon$ について，$p(u)$ が成立するならば，各 $\tau \in \mathcal{T}$ について，$p(\tau(u))$ が成立することを示す．

2.2 代数仕様言語 CafeOBJ

CafeOBJ [1][11][20] は，主に始代数 (*initial algebra*) と隠蔽代数 (*hidden algebra*)[14] を数学的基盤とした仕様記述言語である．始代数は抽象データ型の記述に，隠蔽代数は抽象機械の記述に用いる．

代数は，台集合と，その上の演算の集合で与えられる．台集合は，ソートで表現する．CafeOBJ では，複数のソートを扱うことができ，それらの間に半順序関係を定義できる．このような代数を，順序ソート代数と呼ぶ．演算は，演算子とランクの組で定義する．ランク (*rank*) は，演算の引数として取るソートの列 (アリティ (*arity*) と呼ぶ) と，演算結果のソート (コアリティ(*co-arity*) と呼ぶ) の組である．アリティが空列である演算を定数と呼ぶ．ソートの集合と，演算の集合の組を指標 (*signature*) と呼ぶ．指標で定義した演算の意味は，等式 (公理) で定義する．以上の定義から得られる，指標と等式の組を代数仕様 (*algebraic specification*) と呼ぶ．

CafeOBJ では，可視ソート (*visible sort*) と隠蔽ソート (*hidden sort*) の 2 種類のソートがある．前者は始代数に基づく抽象データ型を，後者は隠蔽代数に基づく抽象機械の状態空間を表すのに用いる．

隠蔽代数では，モデル化の対象となるシステムをブラックボックスとみなし，その外側から観察できる情報だけに基づいて議論を行う．これによって，内部の実装から離れ，抽象度の高いモデル化が行えるという利点を持つ．隠蔽ソート上の演算は，操作演算 (*action operation*)，観測演算 (*observation operation*)，隠蔽定数 (*hidden constant*) の 3 つに分けられる．操作演算は，隠蔽ソートと 0 個以上の可視ソートからなるアリティと，そのアリティに現れる隠蔽ソートをコアリティに持つ演算であり，抽象機械の状態を変化させるのに用いる．観測演算は，隠蔽ソートと 0 個以上の可視ソートからなるアリティと，可視ソートをコアリティに持つ演算であり，抽象機械の様子を観察するのに用いる．

隠蔽定数は，空列のアリティと隠蔽ソートからなるコアリティを持つ演算であり，初期状態の表現などに用いる．抽象機械の振舞いは，操作演算によって観測演算の返戻値がどのように変化するのかを，等式を用いて定義する．

CafeOBJ では，仕様をモジュール単位で記述する．モジュールは，定義済みの別のモジュールを輸入することができる．CafeOBJ 処理系 [1] は，いくつかの定義済みモジュールを内蔵しており，それらはユーザが定義したモジュールと同様に扱える．

このように定義した代数仕様は，各等式を左辺から右辺への書換え規則とみなすことで，項書換え系 [4] として捉えることができる．CafeOBJ は記述した仕様を項書換え系みなし，簡約によって仕様を記号実行することができる．仕様がある性質を持つことを検証したい際には，後述する証明譜を記述し，それを CafeOBJ に実行させ，項を簡約させることで，証明を行う．

2.2.1 CafeOBJ の構文

本節では，CafeOBJ の各構文について，簡潔に記述する．以下では，予約語をタイプライタ書体で表記する．また，省略可能な構文要素を，‘[’ と ‘]’ で括って表記する．説明の都合上，名前を付ける構文要素は，イタリック書体で名前を表記し，‘<’ と ‘>’ で括って表記する．なお，仕様の例として，Standard ML 風のリストを記述した仕様を図 2.1 に示す．図 2.1 では，`nil` は，空のリストを，`::`はリストの構成子を，`@`はリスト同士の連結を，`=`はリスト同士の等価関係を意味する仕様を定義している．

モジュール宣言

モジュールは，キーワード ‘`mod`’ を用いて定義する．

`mod` <*module-name*> [(<*parameters*>)] {
    <*module-elements*>
}

モジュール名を <*module-name*> に，モジュールの各構成要素を <*module-elements*> に記述する．モジュールの構成要素は，後述する輸入，ソートおよびソートの順序関係，演算子，変数および等式に関する各宣言である．記述した代数の意味論をモジュール単位で指定することができる．きつい意味論に基づいてモジュールを記述する場合は，キーワード ‘`mod!`’ を，ゆるい意味論に基づいてモジュールを記述する場合は，キーワード ‘`mod*`’ を，‘`mod`’ に変えて用いる．なお，‘`mod`’ は意味論を指定しない場合に用いる．

モジュールには，パラメータを持たせることができ，<*parameters*> で指定する．パラ

```
mod* TRIV {
  [ Elt ]
}

mod! BASIC-LIST (X :: TRIV) {
  [ NeList < List ]
  op  nil   : -> List
  op  _::_  : Elt List -> NeList
}

mod! LIST-LIB (X :: TRIV) {
  ex (BASIC-LIST (X))
  op  _@_  : List List -> List
  var  E     : Elt
  vars L1 L2 : List
  eq  nil       @ L2 = L2 .
  eq  (E :: L1) @ L2 = E :: (L1 @ L2) .
}

mod! LIST (X :: TRIV) {
  ex (LIST-LIB (X))
  op  _=_  : List List -> Bool  { comm }
  var  L1 : List
  eq  (L1 = L1) = true .
}
```

図 2.1: リストを記述した CafeOBJ の仕様

メータは，仮変数 <*variable-name*> に続いて，‘::’ ，および <*module-name*> で指定する．<*variable-name*> には，<*module-name*> で指定したモジュールに代表されるモジュールを束縛できる．パラメータに実際のモジュールを束縛することを，モジュールの具象化と呼ぶ．

図 2.1 中では，きつい意味論に基づくモジュール BASIC-LIST，LIST-LIB および LIST を，ゆるい意味論に基づくモジュール TRIV を宣言している．また，BASIC-LIST，LIST-LIB および LIST は，TRIV[*2] に代表されるモジュールをパラメータとしている．

輸入宣言

輸入宣言は，キーワード ‘pr’ ，‘ex’ および ‘us’ を用いて行う．

pr(<*module-exp*>)

‘ex’ や ‘us’ も同様の構文である．<*module-exp*> は，モジュール式である．輸入するモジュール名だけからなる式 <*module-name*> は，モジュール式であり，<*module-name*>

[*2] TRIV は，CafeOBJ の組込みモジュールであり，実際には改めて定義する必要はない．

で指定したモジュールを輸入することを意味する．2 つのモジュール式を ‘+’ で結んだ式もまた，モジュール式である．これは，‘+’ で結ばれた両辺に指定された各モジュール式に書かれたモジュールが輸入されることを意味する．例えば，NAT + STRING は，モジュール NAT と STRING を輸入することを意味する式である．$<$*module-name*$>$ * (...) もまた，モジュール式であり，ソートや演算子の名前替えや，モジュールが持つパラメータの具象化など，様々な指定ができる．しかし，ここでは，それらの構文の詳細については触れず，必要の都度自然語で解説を加える．

モジュール M において，‘pr(M')’，‘ex(M')’ または ‘us(M')’ が記述されると，それぞれ次の意味を持つ．

- pr : M の宣言は，M' で宣言されているソートをコアリティに持つ，新しい演算を加えない．かつ，M' で等しくない項を，等号で結ばない．
- ex : M の宣言は，M' で等しくない項を，等号で結ばない．
- us : M の宣言は，何を行ってもよい．

すべてのモジュールにおいて，組込みモジュール BOOL が暗黙に輸入される．BOOL では，真偽値と論理演算が定義されている．

図 2.1 中では，LIST-LIB において BASIC-LIST を，LIST において LIST-LIB を輸入している．ここで，被輸入モジュールは，パラメータを持っており，それらも同じモジュールで具象化するため，自身の仮引数を被輸入モジュールに渡している．例えば，自然数を定義している組込みモジュール NAT を用いて LIST を具象化すると，被輸入モジュールである LIST-LIB と BASIC-LIST のパラメータにも，それぞれ NAT が渡され，自然数のリストが利用可能になる．

ソート宣言

可視ソート V はキーワード ‘[’ と ‘]’ で，隠蔽ソート H はキーワード ‘*[’ と ‘]*’ で括って宣言する．

[V]

[H]

また，ソート V と V' の間の順序関係は，キーワード ‘<’ を用いて次のように宣言する．

[V' < V]

このとき，V' は V のサブソート (または V は V' のスーパーソート) であると呼ぶ．順序ソートは，隠蔽ソート H と H' についても同様に定義できる．ただし，循環する順序

関係および，可視ソートと隠蔽ソートの間の順序関係は定義できない．なお，複数のソートや，それらの間の順序関係を同時に定義するための構文もあるが，複数の宣言を繰り返すことで同等の効果を得られるため，割愛する．

図 2.1 中では，TRIV において，可視ソート Elt を宣言している．BASIC-LIST において，可視ソート List とそのサブソート NeList を定義している．

演算子宣言

演算子宣言は，キーワード ‘op’ を用いる．操作演算と観測演算の宣言には，キーワード ‘bop’ を用いる．‘op’ や ‘bop’ に続いて，演算子名を記述し，‘:’，アリティの列，‘->’，およびコアリティを記述する．アリティが n 個の可視ソート $V_1, \ldots, V_n$ の列からなり，コアリティが可視ソート V である演算子 f は，次のように宣言する．

op f : V_1 V_2 ... V_n -> V

同じランクを持つ複数の演算は，キーワード ‘ops’ や ‘bops’ によって，まとめて宣言できる．‘ops’ や ‘bops’ に続く構文は，‘op’ や ‘bop’ と同じである． アリティが n 個の可視ソート $V_1, \ldots, V_n$ の列からなり，コアリティが可視ソート V である， m 個の演算子 $f_1, f_2, \ldots, f_m$ は，次のように宣言する．

ops f_1 f_2 ... f_m : V_1 V_2 ... V_n -> V

演算子名に ‘_’ (下線) を入れることで，アリティの位置を指定できる．例えば，図 2.1 中では，::を中置演算子として宣言している．

演算子には，その演算に成り立つ性質 (結合律，交換律など) を，属性として宣言できる．属性は，演算の宣言に続いて，‘{’ と ‘}’ で括って宣言する．

op f : V_1 V_2 ... V_n -> V { <*attribute*> <*attribute*> ... }

<*attribute*> に指定できる，代表的な演算子の属性を以下に示す．

- assoc : 演算 f に結合律が成り立つ．
- comm : 演算 f に交換律が成り立つ．
- idem : 演算 f にべき等律が成り立つ．
- prec: <*precedence*> : 構文解析の際の，演算 f の優先順位 <*precedence*> を自然数で指定する．数値が小さい演算が優先順位が高い．
- strat (<*e-strategy*>) : 簡約の際の，演算の評価戦略として，e 戦略を採用する．<*e-strategy*> には，自然数の列が入り，例えば 1 2 0 の場合は，演算のアリティに現れる第一引数，第二引数，項全体の順で簡約される．

図 2.1 中では，定数 `nil`，二項演算子 `_::_`，`_@_`，`_=_` を宣言している．また，`_=_` には，交換律が成り立つことを宣言している．

変数宣言

変数は，等式で用いられ，指定したソートの任意の項が代入できる．変数は，キーワード ‘`var`’ で宣言する．‘`var`’ に続いて，変数名，`:` および 変数のソートの順に記述する．同じソート上の変数は，キーワード ‘`vars`’ でまとめて宣言できる．ソート V 上の変数 v (または V 上の n 個の変数 $v_1, v_2, \ldots, v_n$) は，次のように記述する．

`var` v `:` V

`vars` v_1 v_2 ... v_n `:` V

図 2.1 中では，可視ソート `Elt` 上の変数として `E` を，`List` 上の変数として，`L1` と `L2` を宣言している．

変数の有効範囲は，変数を宣言したモジュール内である．

等式宣言

等式は，二つの項 <*lhs*> と <*rhs*> の間の等価関係を宣言する．等式は，演算の意味を定義するために用いる．等式の宣言はキーワード ‘`eq`’ に続いて，左辺の項 <*lhs*>，‘`=`’，右辺の項 <*rhs*> および ‘`.`’ の順に記述することで行う．

`eq` <*lhs*> `=` <*rhs*> `.`

条件付き等式は，キーワード ‘`ceq`’ を用いて宣言する．条件 <*condition*> は，CafeOBJ 組み込みのソート `Bool` 上の項であり，‘`if`’ に続いて記述し，最後に ‘`.`’ を記述する．

`ceq` <*lhs*> `=` <*rhs*> `if` <*condition*> `.`

図 2.1 中では，`@` (リスト同士の結合) を定義するために，2 つの等式が，`=` (リスト同士の等価関係) を定義するために，1 つの等式が宣言されている．

また，等式の左辺 <*lhs*> の中で，変数を宣言することもできる．構文 $v : V$ は，等式中において，ソート V 上の変数 v を宣言している．このようにして宣言した変数は，その等式中においてのみ有効である．

2.2.2 CafeOBJ の証明譜

本節では，証明譜の構文について述べる．証明譜は，CafeOBJ で記述した仕様がある性質を満たすことを証明したいときに記述する．実際の証明譜の例を，図 2.2 に示す．こ

```
open LIST
ops l1 l2 l3 : -> List .
op  e : -> Elt .
-- proof the associativity of _@_ by induction.
-- base step.
red (nil @ l2) @ l3 = nil @ (l2 @ l3) .
-- induction hypothesis.
eq (l1 @ l2) @ l3 = l1 @ (l2 @ l3) .
-- induction step.
red ((e :: l1) @ l2) @ l3 = (e :: l1) @ (l2 @ l3) .
-- Q.E.D.
close
```

図 2.2: 演算 _@_ の結語律を示すための証明譜

れは，前節で使用したリストの仕様を用い，@ に結合律が成り立つことを，帰納法を用いて検証するための証明譜である．

検証もまた，モジュール単位で行う．あるモジュールにおいて，ある性質が成り立つことを証明することを考える．CafeOBJ の検証は，証明したい性質を表す項 (述語) を簡約することで進める．しかし，一般には証明したい性質一つに対して，一度の簡約で済むとは限らない．その最大の理由は場合分けであり，検証の際には，それを目的として，既にあるモジュールに新たな演算や等式を加えることがある．このため，場合分け毎に，定数や演算を加え，述語を簡約し，場合分けのために加えた定数や演算を取り除くという作業を繰り返すことになる．CafeOBJ では，モジュールをオープンすることで，検証に必要な定数や演算を一時的に加え，クローズすることで，それらを取り除き，元の定義に戻すことができる．これらの操作はキーワード ‘open’ と ‘close’ で行う．

open <*module-name*>
    <*proof-elements*> .
close

<*proof-elements*> には，前述した演算，変数，および等式の各宣言 (ただし，前述した構文に加えて，終端に ‘.’ が必要である) に加えて，簡約を行うコマンド ‘red’ を記述できる．項 <*term*> を簡約するには，次のように記述する．

red <*term*> .

以下では，図 2.2 に示した証明譜を用い，@ に結合律が成り立つことを，リストの長さに関する帰納法を用いて証明する．検証の対象となるモジュールは LIST である．LIST はパラメータを持っているが，@ の結合律は，パラメータに束縛するモジュールに依存せずに成り立つ性質であるので，具象化せずに検証を行う．

```
CafeOBJ> in list
processing input : ./list.mod
-- defining module! BASIC-LIST_*_*...._* done.
-- defining module! LIST-LIB_*_*.,,,,,,,*....._...* done.
-- opening module LIST-LIB(X).. done._*
-- reduce in %LIST-LIB(X) : ((nil @ l2) @ l3) = (nil @ (l2 @ l3))
true : Bool
(0.010 sec for parse, 3 rewrites(0.000 sec), 11 matches)
*
-- reduce in %LIST-LIB(X) : (((e :: l1) @ l2) @ l3)
                          = ((e :: l1) @ (l2 @ l3))
true : Bool
(0.000 sec for parse, 5 rewrites(0.000 sec), 26 matches)
CafeOBJ>
```

図 2.3: 演算 _@_ の結語律を検証した実行結果

証明　$\forall l_1.\forall l_2.\forall l_3.(l_1 @ l_2) @ l_3 = l_1 @ (l_2 @ l_3)$

最初に，検証の対象となるモジュールをオープンする．次に，$\forall l_1$，$\forall l_2$ および $\forall l_3$ を表現するために，List 上の定数 l1，l2 および l3 を宣言する．同様に，Elt 上の定数として e を宣言する．

ここでは，l_1 を帰納法の変数とする．従って，基底では，次の項を簡約することで，基底が成り立つことを示す．

```
(nil @ l2) @ l3 = nil @ (l2 @ l3)
```

次に，帰納法の仮定として，等式 (l1 @ l2) @ l3 = l1 @ (l2 @ l3) を宣言する．e :: l1 は，l1 より 1 だけ長いリストである．よって，帰納段階では，次の項を簡約することで，基底が成り立つことを示す．

```
(e ::  l1 @ l2) @ l3 = e ::  l1 @ (l2 @ l3)
```

図 2.2の証明譜を CafeOBJ の処理系に読み込ませ，実行した様子を 図 2.3に示す．証明譜に記述した 2 つの簡約結果はいずれも true であり，検証が成功したことがわかる．□

上述したように，CafeOBJ の検証は，等式を左辺から右辺へ書き換えることにより，機械的に進む．このことは，等式推論による正しさを保証したまま，迅速な検証を可能にしている．

第 3 章

CafeOBJ による振舞遷移機械の記述

本章では，振舞遷移機械を CafeOBJ で記述する手順について，例を示しながら述べる．例には，簡易な鉄道信号システムであるスタフ閉そくを用いる．振舞遷移機械は，CafeOBJ の振舞仕様として記述されることが多い．最初に，3.1 節で，スタフ閉そくがどのようなシステムであるかを述べ，3.2 節で記述の手順について述べる．この手順と例題は，筆者が [35] で示したものである．

3.1 スタフ閉そく

本節では，例題として使用するスタフ閉そくが，どのようなシステムであるか解説し，それを振舞遷移機械で表したモデルについて述べる．

3.1.1 概要

スタフ閉そく [2][3] とは，単線区間に適用される信号システムである．このシステムは，隣接する二つの停車場間を結ぶ単線区間において，列車同士の正面衝突および追突を防ぐことを目的とする．停車場とは，列車の交換設備を有する駅のことである．図 3.1は，スタフ閉そくを適用した鉄道システムから，互いに隣接する任意の二つの停車場と，それらを結ぶ単線の線路を抜き出して，示している．図中の記号の意味は以下の通りである．

- a および b : 停車場の識別子．
- tn $(n = 1, \ldots, 7, \mathsf{l}, \mathsf{r})$: 線路の識別子．線路は，区間単位に区切って表現する．tl

は停車場 a より左側の線路 (または車庫)，tr は b より右側の線路 (または車庫) である．t1(または t7) は，駅 a(または駅 b) から右 (または左) 方向へ走る列車が発着する区間，t2(または t6) は，駅 a(または駅 b) から左 (または右) 方向へ走る列車が発着する区間である．t3 と t5 は，線路が分岐している区間である．t4 はこれらの駅を結ぶ単線区間である．

図 3.1 には表記していないが，他に列車とスタフがある．各列車は識別子と運転方向 l または r を持ち，区間を単位として移動する．運転方向が r (図 3.1 において，左から右に運転される) である列車は，初期状態では，tl に存在し，t1 ，t3 ，t4 ，t5 ，t6，tr の順に移動する．逆に，運転方向が l である列車は，tr，t7 ，t5 ，t4 ，t3 ，t2，tl の順で移動する．

スタフは，停車場 a と b 間において，唯一の物体である．初期状態では，スタフはどちらかの停車場に存在する．停車場 a (または b) がスタフを所有するとき，t1 (または t7) にいる列車にスタフを渡すことができる．スタフを受け取った列車は，その停車場 a (または b) から発車し，単線の区間 t4 に進入し，隣の停車場の区間 t6 (または t2) まで移動できる．列車が t6 (または t2) に到着すると，停車場 b (または a) へスタフを返却する．列車は，そのスタフを持ったまま，tr (または tl) へ移動してはならない．

以上をまとめると，主要なルールは，次の 4 つである:

- 列車 c がスタフを所有しているとき，c は区間 t1 または t7 から移動できる．
- 列車 c がスタフを所有していないとき，c は区間 t2 または t6 から移動できる．
- 列車 c は，c が区間 t1 (または t7) におり，停車場 a (または b) がスタフを所有するとき，スタフを受領できる．
- 列車 c は，c が区間 t1 または t2 (または t6 または t7) におり，かつ c がスタフを所有するとき，停車場 a (または b) へスタフを返却できる．

このようなルールに従うとき，スタフ閉そくを施行する停車場 a と b において，区間 t4 に同時に 2 列車が進入することはない，つまり区間 t4 において，列車同士の衝突は起こらないことが保証できる．

3.1.2 スタフ閉そくのモデル

ここでは，前述したスタフ閉そくを，振舞遷移機械でモデル化する．

最初に，データとして，区間，列車および駅の識別子の各集合 T，C および Z と，方向の値の集合 D を，次のように定義する．

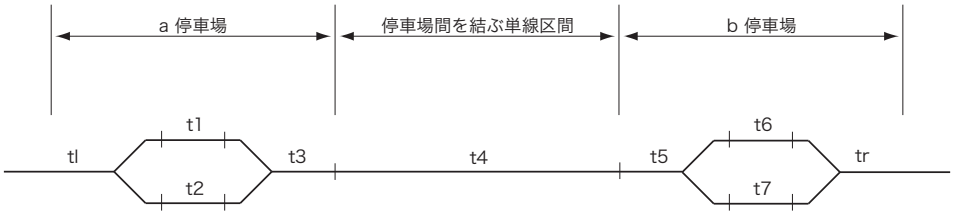


図 3.1: スタフ閉そくを適用した鉄道システム

- $T = \{\mathsf{tl}, \mathsf{t1}, \mathsf{t2}, \mathsf{t3}, \mathsf{t4}, \mathsf{t5}, \mathsf{t6}, \mathsf{t7}, \mathsf{tr}\}$. 区間の識別子.
- $C = \{0, 1, 2, \ldots\}$. 列車の識別子.
- $D = \{\mathsf{l}, \mathsf{r}\}$. 列車の進行方向を表す値.
- $Z = \{\mathsf{a}, \mathsf{b}\}$. 駅の識別子.

これらのデータを用いて，スタフ閉そくを表す振舞遷移機械 $\mathcal{S} = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ を定義する．なお，2.1 で述べたように，スタフ閉そくを表す状態空間 Υ の存在を仮定する．

観測 $\mathcal{O}$ は次のように表現できる．

- $\{pos_c : \Upsilon \to T \mid c \in C\}$. 列車 c の現在位置を区間の識別子 $t \in T$ で観測する.
- $\{dir_c : \Upsilon \to D \mid c \in C\}$. 列車 c の進行方向を方向の値 $d \in D$ で観測する.
- $staff : \Upsilon \to (C \cup Z)$. スタフの所有者を観測する．スタフの所有者は，いずれかの停車場か，いずれかの列車であるので，それらの和集合 $C \cup Z$ の元で表現できる.

初期状態 $\mathcal{I}$ は，任意の状態 $u \in \Upsilon$ に対し，次のように表現できる.

- $\forall c \in C.(dir_c(u) = \mathsf{r} \bigtriangledown dir_c(u) = \mathsf{l})$. 列車は右方向または左方向へ走行する.
- $\forall c \in C.(dir_c(u) = \mathsf{r} \Rightarrow pos_c(u) = \mathsf{tl})$. 右方向へ走行する列車は，tl にいる.
- $\forall c \in C.(dir_c(u) = \mathsf{l} \Rightarrow pos_c(u) = \mathsf{tr})$. 左方向へ走行する列車は，tr にいる.
- $staff(u) = \mathsf{a} \bigtriangledown staff(u) = \mathsf{b}$. スタフはどちらかの停車場が所有している.

ただし，$\bigtriangledown$ は排他的論理和とする.

遷移規則 $\mathcal{T}$ は，3 つの添字付き遷移規則の集合で表現する．ここでは，遷移規則が適用された時の状態を $u \in \Upsilon$，遷移規則が適用された後の状態を $u' \in \Upsilon$ と表記する.

- $\{move_{(c,t,d)} : \Upsilon \to \Upsilon \mid c \in C, t \in T, d \in D\}$. 列車 c が区間 t から方向 d に隣接する区間へ移動することを表す遷移規則の集合である.
  効力条件は，$pos_c(u) = t \wedge dir_c(u) = d$ である．これは，列車 c が区間 t から方向 d に隣接する区間へ移動するためには，c は状態 u において，実際に区間 t に存

在し，方向 d へ走行していなければならないことを意味している．ただし，t1 (または t7) から r (または l) へ移動するには，スタフを所有していることが条件として加わるので，前述の条件に $\wedge \mathit{staff}(u) = c$ を加えることで表現する．同様に，t6 (または t2) から r (または l) へ移動するには，スタフを返却していることが条件として加わるので，前述の条件に $\wedge \mathit{staff}(u) \neq c$ を加えることで表現する．また，一方通行の区間 t2 (または t6) から r (または l) へは移動できないので，これを表す操作演算の効力条件は偽と定義することで表現する．

効果は，$pos_c(u')$ が，方向 d に隣接する区間の識別子に変化することである．

- $\{catch_{(c,z)} : \Upsilon \rightarrow \Upsilon \mid c \in C, z \in Z\}$． 列車 c が停車場 z からスタフを受領することを表す遷移規則の集合である．

  $catch_{(c,\mathsf{a})}(u)$ の効力条件は，$pos_c(u) = \mathsf{t1} \wedge \mathit{staff}(u) = \mathsf{a}$ である．これは，列車 c が区間 t1 に存在し， 停車場 a がスタフを所有するとき，列車 c が停車場 a からスタフを受領できることを意味している．$catch_{(c,\mathsf{b})}(u)$ も同様に定義できる．

  $catch_{(c,\mathsf{a})}(u)$ と $catch_{(c,\mathsf{b})}(u)$ の効果は，$\mathit{staff}(u') = c$ である．これは，スタフの所有者が c となることを意味している．

- $\{release_{(c,z)} : \Sigma \rightarrow \Sigma \mid c \in C, z \in Z\}$． 列車 c が停車場 z へスタフを返却することを表す遷移規則の集合である．

  $release_{(c,\mathsf{a})}(u)$ の効力条件は，$(pos_c(u) = \mathsf{t1} \vee pos_c(u) = \mathsf{t2}) \wedge \mathit{staff}(u) = c$ である．これは，列車 c が区間 t1 または t2 に存在し， その列車がスタフを所有するとき，列車 c が停車場 a へスタフを返却できることを意味している．$release_{(c,\mathsf{b})}(u)$ についても，同様に定義できる．

  $release_{(c,\mathsf{a})}(u)$ の効果は，$\mathit{staff}(u') = \mathsf{a}$ である．これは，スタフの所有者が a となることを意味している．$release_{(c,\mathsf{a})}(u)$ についても，同様に定義できる．

3.2 記述の手順

本節では，CafeOBJ の仕様をどのように記述するかについて述べる．例題として，前述したスタフ閉そくを用いる．完全なコードは，付録 A.1 と A.2 に示す．

3.2.1 抽象データ型の記述

抽象機械の記述に先立って，データを抽象データ型として記述する．CafeOBJ は，その内蔵モジュールに，抽象データ型を表現したものを有している．例えば，モジュール

```
mod! STATIONID {
  [ StID ]
  ops a b : -> StID
  op  _=_ : StID StID -> Bool  { comm }
  eq (ST:StID = ST) = true .
  eq (a = b) = false .
}
```

図 3.2: モジュール STATIONID

BOOL，NAT，INT は，それぞれ真偽値や論理演算，自然数，整数を定義している．他に，列挙型やより複雑なデータ型 (集合，スタックなど) も既知 [11] であるため，その記述法については割愛する．

手順 3.1. 等価性

抽象データ型を表す可視ソート V 上の項 v_1, v_2 について，等価性を判定する際には注意を要する．記述した抽象データ型が，項書換え系として合流性を持つならば，CafeOBJ の組込み述語==を用いて，完全な等価性を判定できる．==は，その両辺の項を簡約し，それが構文的に同じ項であるとき真を，そうでないときに偽を返す述語である．しかし，本論文では，次の 2 つの理由から，==は使用しない．第一に，合流性を保ちながら抽象データ型を記述することは困難である．第二に，合流性を有する抽象データ型の仕様を用い，抽象機械を記述したとしても，抽象機械の記述において不用意に==を用いると，検証の健全性を崩すことが知られている．そこで，本論文では健全な等価性を判定する述語=を宣言し，これを用いて等価性を判定する．=は，等しいか，または等しくないことが既知である項に対してのみ，真または偽を返すように定義する．典型的には，次のように定義する．

```
op _=_ : V V -> Bool { comm }
eq (v:V = v) = true .
```

v は V 上の変数である．ここでは，= は，その両辺の項をそれぞれ簡約し，それらが構文的に同じ項であるとき，真を返す．

例 3.1. スタフ閉そくで使用する，集合 Z を，図 3.2 に示すように記述した．

集合 Z は列挙型であり，要素 a と b を有する．これを可視ソート StID と，その上の定数 a と b として表現した．さらに，これらの間の等価性を判定するための述語= と，その意味を与える等式を宣言している．最初の等式は，=の両辺にある項をそれぞれ簡約し，構文的に同じであるならば，等しいと判定することを宣言している．次の等式は，a と b は異なる要素であることを宣言している．

```
mod* STAFFSYSTEM {
  pr(TCID + TRDIR + STAFFPOS)                    ┐ (1) 輸入宣言および
  *[ State ]*                                    ┘     隠蔽ソートの宣言

  bop pos : TrID State -> TcID                   ┐
  bop dir : TrID State -> TrDir                  │ (2) 観測演算の宣言
  bop staff : State -> StaffPos                  ┘

  op  init  : -> State                           ┐
  ceq pos(TR:TrID, init) = tl                    │ (3) 初期状態の宣言
      if  dir(TR, init) = r .                    │
  ceq pos(TR:TrID, init) = tr                    │
      if  dir(TR, init) = l .                    ┘

  ...
}
```

図 3.3: モジュール STAFFSYSTEM

スタフ閉そくでは，集合 T ，C，D を使用するので，これらを可視ソート TrID，TrID，TrDir と，それらの上の演算によって表現した．また，集合 $C \cup Z$ を表現するために，C と Z をサブソートとする可視ソート StaffPos を宣言した．

3.2.2 抽象機械の記述

データの集合 $D, D_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ と，$\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ で表される任意の振舞遷移機械 $\mathcal{S}$ を CafeOBJ で記述するには，以下の手順 2 から 5 に沿って仕様を記述する．なお，ここでは各データの集合 $D, D_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ が，可視ソート $V, V_k(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$ 上に適切に記述されていることを仮定する．

手順 3.2. 準備

振舞遷移機械 $\mathcal{S}$ を定義するモジュールを宣言する．そして，記述に使用するデータが定義されている各モジュールを輸入し，状態空間 Υ を表す隠蔽ソート H を宣言する．

例 3.2. スタフ閉そくを表すモジュール STAFFSYSTEM は図 3.3に示すように記述できる．

スタフ閉そくを表す抽象機械をモジュール STAFFSYSTEM に記述した．その中で，データを定義したモジュール TCID，TRDIR，STAFFPOS を輸入し，Υ を表す隠蔽ソート H を宣言した．これを図 3.3 (1) に示す．

手順 3.3. 観測

振舞遷移機械の定義に従って，各観測 $o \in \mathcal{O}$ を表す CafeOBJ の観測演算を宣言する．添字によってまとめられた観測 $o_{(i_1,\ldots,i_m)}$ について，観測演算 o を一つ定義し，添字

$i_1, \ldots, i_m$ は観測演算のアリティで表現する．つまり，各観測 $o_{i_1,\ldots,i_m} : \Upsilon \to D$ について，観測演算 o を，以下に示すように宣言する．

`bop` o `:` V_{i_1} $\ldots$ V_{i_m} H `->` V

例 3.3. スタフ閉そくの観測は，3 つの観測演算で表現できる．これを図 3.3 (2) に示す．

手順 3.4. 初期条件

任意の初期状態 $u \in \Upsilon$ について，H 上の隠蔽定数 $init$ を，以下に示すように宣言する．

`op` $init$ `: ->` V

次に，各観測 $o_{(i_1,\ldots,i_m)}$ の初期値を，等式で記述する．u を各 $o_{(i_1,\ldots,i_m)}$ で観測した値が $f(i_1, \ldots, i_m)$ で表されるとき，次の等式で表現できる．

`eq` o`(`v_{i_1}`:`V_{i_1}`,` $\ldots$`,` v_{i_m}`:`V_{i_m}`,` $init$`) = `f`(`v_{i_1}`,` $\ldots$`,` v_{i_m}`)` .

例 3.4. スタフ閉そくの初期条件は，図 3.3 (3) に示すように表現できる．

図 3.3 (3) の `init` は，初期状態を表す隠蔽定数である．2 つの条件付き等式は，観測演算 `pos` の初期値を宣言している．観測演算 `dir` および `staff` の初期値は，それぞれ `TrDir` と `StaffPos` 上の任意の項であるため，それらの初期値を宣言しない．

手順 3.5. 遷移規則

振舞遷移機械の定義に従って，各遷移規則 $\tau \in \mathcal{T}$ を表す CafeOBJ の操作演算を宣言する．添字部分は，観測演算と同様に操作演算のアリティで表現する．つまり，各遷移規則 $\tau_{j_1,\ldots,j_n} : \Upsilon \to \Upsilon$ について，操作演算 τ を，以下に示すように宣言する．

`bop` τ `:` V_{j_1} $\ldots$ V_{j_n} H `->` H

振舞遷移機械の各遷移規則は，効力条件と効果の組で定義されている．この定義と CafeOBJ の記述の親和性を高めるため，CafeOBJ でも各 $\tau \in \mathcal{T}$ の効力条件を表す演算 c_τ と，各 τ が各 $o \in \mathcal{O}$ に及ぼす効果を表す演算 $e_{(\tau,o)}$ を宣言し，これらを用いて表現する．これらの演算のランクは，操作演算 τ と観測演算 o のランクから機械的に決まる．この規則を以下に示す．

- 操作演算 τ `:` V_{j_1} $\ldots$ V_{j_n} H `->` H の効力条件を表す演算 c_τ は，τ のアリティから，次のように宣言する．

  `op` c_τ `:` V_{j_1} $\ldots$ V_{j_n} H `-> Bool`

- 操作演算 τ `:` V_{j_1} $\ldots$ V_{j_n} H `->` H の，観測演算 o `:` V_{i_1} $\ldots$ V_{i_m} H `->` V の効果を表す演算 $e_{(\tau,o)}$ は，τ と o のランクから，次のように宣言する．

  `op` $e_{(\tau,o)}$ `:` V_{i_1} $\ldots$ V_{i_m} V_{j_1} $\ldots$ V_{j_n} H `->` V

これらの演算を用いて，効力条件と効果を次のように等式で記述する．各操作演算 τ の効力条件が，$cond_\tau : D_{j_1} \dots D_{j_n} \to \{\text{true}, \text{false}\}$ で与えられるとき，次の等式を宣言する．

$$\texttt{eq}\ c_\tau(v_{j_1}{:}V_{j_1}, \dots, v_{j_n}{:}V_{j_n}, h{:}H)\ =\ cond_\tau(v_{j_1}, \dots, v_{j_m}, h)\ .$$

さらに，操作演算 τ が観測演算 o に及ぼす効果が，$\mathit{eff}_{(\tau,o)} : D_{i_1} \dots D_{i_m} D_{j_1} \dots D_{j_n} \to D$ で与えられるとき，次の等式を宣言する．

$$\begin{aligned}&\texttt{eq}\ e_{(\tau,o)}(v_{i_1}{:}V_{i_1}, \dots, v_{i_m}{:}V_{i_m}, v_{j_1}{:}V_{j_1}, \dots, v_{j_n}{:}V_{j_n}, h{:}H)\\ &\qquad =\ \mathit{eff}_{(\tau,o)}(v_{i_1}, \dots, v_{i_m}, v_{j_1}, \dots, v_{j_n}, h)\ .\end{aligned}$$

最後に，状態機械の振舞いを表現するための等式を宣言する．観測演算 o と 遷移規則 τ の各組について，$e_{(\tau,o)}$ と c_τ を用いて，次の 2 つの等式を記述する．

$$\begin{aligned}&\texttt{ceq}\ o(v_{i_1}{:}V_{i_1}, \dots, v_{i_m}{:}V_{i_m}, \tau(v_{j_1}{:}V_{j_1}, \dots, v_{j_n}{:}V_{j_n}, h{:}H))\\ &\qquad =\ e_{(\tau,o)}(v_{i_1}, \dots, v_{i_m}, v_{j_1}, \dots, v_{j_n}, h)\\ &\quad \texttt{if}\ c_\tau(v_{j_1}, \dots, v_{j_n}, h)\ .\\ &\texttt{ceq}\ o(v_{i_1}{:}V_{i_1}, \dots, v_{i_m}{:}V_{i_m}, \tau(v_{j_1}{:}V_{j_1}, \dots, v_{j_n}{:}V_{j_n}, h{:}H))\\ &\qquad =\ o(v_{i_1}, \dots, v_{i_m}, h)\\ &\quad \texttt{if}\ \neg c_\tau(v_{j_1}, \dots, v_{j_n}, h)\ .\end{aligned}$$

1 つ目の等式は，操作演算 τ が適用されたとき，効力条件 c_τ が真ならば，効果 $e_{(\tau,o)}$ によって，観測演算 o が変化することを表現している．2 つ目の等式は，c_τ が偽ならば，観測演算 o が変化しないことを表現している．

以上の手順により，振舞遷移機械 $\mathcal{S}$ から CafeOBJ の仕様を記述できる．

例 3.5. スタフ閉そくの遷移規則は，図 3.4 に示す記述で表現した．

スタフ閉そくの各遷移規則を，図 3.4 (1) に示す 3 つの操作演算 `move`，`catch` および `release` で表現した．次に，各操作演算について，効力条件と効果を記述する．しかし，すべての例を見るのは冗長であるので，ここでは，遷移規則 $move_{(c,\mathsf{t1},\mathsf{r})}$ だけを取り上げる．他の遷移規則についても同様に定義できる．

遷移規則 $move_{(c,\mathsf{t1},\mathsf{r})}$ の効力条件と効果の定義を，以下に示す．

効力条件: $pos_c(u) = \mathsf{t1} \wedge dir_c(u) = \mathsf{r} \wedge \mathit{staff}(u) = c$.

効果: $pos_c(u') = \mathsf{t3}$

$move_{(c,\mathsf{t1},\mathsf{r})}$ は，列車 c が区間 t1 におり，c は r 方向へ走っていて，かつ c がスタフを持っているとき，c の位置が t3 に変わる (他の観測は変わらない) ことを意味している．

操作演算 `move` の効力条件を表す演算 `c-move` ，操作演算 `move` が各観測演算に及ぼす効果を表す演算 `e-pos-move`，`e-dir-move` および `e-staff-move` の記述を，図 3.4 (2)

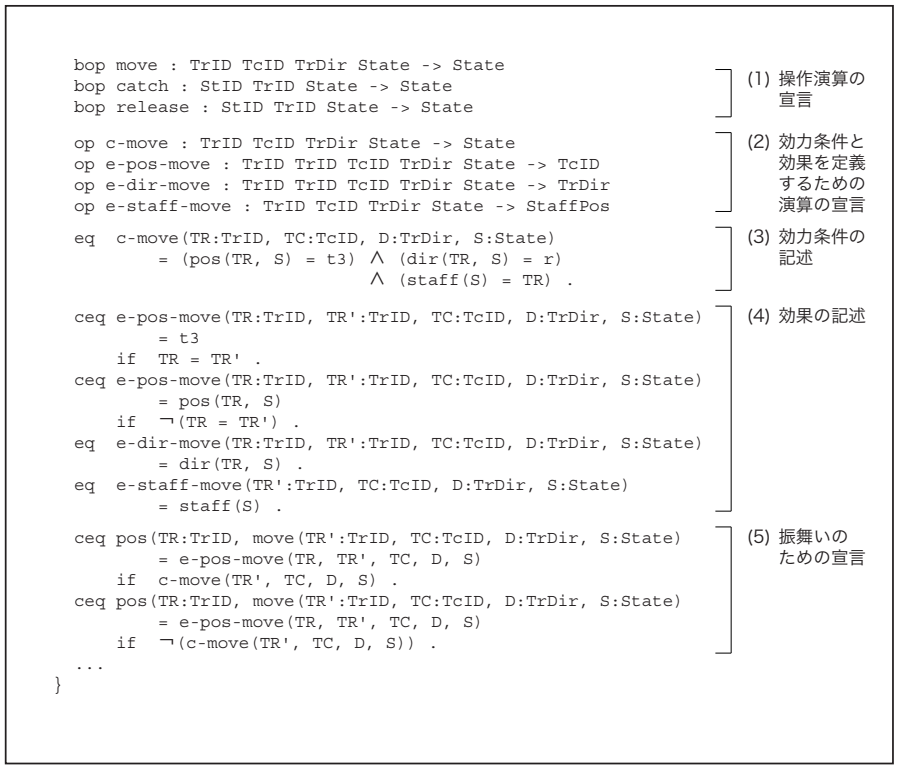


図 3.4: モジュール STAFFSYSTEM

に示す．図 3.4 (3) は，c-move を用いて効力条件を記述した等式である．図 3.4 (4) は，e-pos-move，e-dir-move および e-staff-move を用いて，効果を記述した等式である．

同様の定義を，他の操作演算についても記述した．

最後に，操作演算の適用により，観測演算がどのように変化するか (振舞い) を記述した．図 3.4 (2) から (4) で宣言した演算を用い，図 3.4 (5) に示す等式を宣言した．操作演算 catch および release についても，同様の等式を宣言した．

以上の手順によって，振舞遷移機械で記述したスタフ閉そくシステムを，CafeOBJ で記述できた．

第 4 章

安全性検証の組織化

1 章で述べたように，本論文において，振舞遷移機械の安全性検証には帰納法を用いる．これまで，CafeOBJ を用いた帰納法による検証は，図 1.1に示すように，適切な場合毎に分けて記述された多数の証明譜の集まりであった．検証者は，個々の証明譜を，その検証の一部を担うように記述し，それらの集まりで，その検証について議論し尽くすように記述しなければならない．このことは，検証における見落としや，論理の飛躍を招くことがあった．

そこで 4 章では，この問題を解決するために，いくつかの手法を提案する．最初に，場合を項で表現する手法を提案する．これによって，次の二つの手法を現在の CafeOBJ 処理系で実現できる．一つ目は，場合分けの網羅性を保証するために，検証に用いる場合分けをマトリクス状に整理する手法である．このマトリクスは，状態機械の仕様に由来する軸と，表明に由来する軸から成り，後者をパラメタとしている．このため，表明毎にマトリクスを再利用できる．二つ目は，前述の手法でうまく検証できないときに，検証を進める上で有効な場合分けや補題を発見するための手法である．この手法では，検証できなった場合を元に，機械的に場合を生成し検証を試みることで，検証を進める手掛かりとなる場合分けの候補を発見する．検証者は，見つかった場合分けを，前述のマトリクスに追加することで，検証を進めることができる．

本章でも 3 章に引き続きスタフ閉そくを例題として使用する．本手法は，筆者が [35] で示した．

4.1　項による場合の表現

4.1.1　隠蔽定数による場合の表現

本論文において，場合分けとは，状態空間に関する適当な述語を使い，状態空間を分割することである．場合とは，分割された各状態空間のことである．

ある条件を満たす任意の状態を表す隠蔽ソートの定数を隠蔽定数と呼ぶ．1 つの隠蔽定数は，1 つの場合を表現している．今，状態に関する n 個の述語 $p_i : \Upsilon \to \{\mathrm{true}, \mathrm{false}\}(i = 1, \ldots, n)$ を考え (ただし，$n = 0$ のときは，場合分けを行わないときを表す)，各 p_i は，次の制限を満たすものとする:

- 述語 p はリテラルに限定する．リテラルとは，原始述語 (論理結合子を含まない論理式) か，原始述語に否定 ($\neg$) が 1 つだけ付いた論理式である．
- p に表れる変数は，定数で具象化されている．

この制限を満たす述語は，$\hat{p}$ のように記述する．すべての $\hat{p}_i(i = 1, \ldots, n)$ が成立する任意の状態を表す隠蔽定数 s_p は，次のように宣言する:

```
op s_p : -> H
eq p̂_1(s_p) = true .
   ⋮
eq p̂_n(s_p) = true .
```

最初に，定数 s_p を宣言し，s_p で表される任意の状態おいて，各 $\hat{p}_i$ が真であることを等式で宣言する．ただし，$n = 0$(場合分けをしない) のときは，等式は宣言されず，s_p は任意の状態を表す．証明譜では，隠蔽定数は場合を表現するために用いられる．例えば，ある性質を検証するために m 通りの場合に分割した議論が必要であるときは，各々の場合を表す隠蔽定数 $s_1, \ldots, s_m$ を宣言する必要がある．

以下では，証明譜を記述し，等式を宣言するのではなく，各場合を項で表現する手法を提案する．最初に，仕様や表明に表れる各原始述語 p について，p に表れる変数を定数で具象化し，$\hat{p}$ を作る．相補対 $(\hat{p}, \neg\hat{p})$ について，隠蔽定数 $s_{\hat{p}}$ と $s_{\neg\hat{p}}$ を宣言する．今，ある場合を表す論理式 p が，$\hat{p}_1 \wedge \ldots \wedge \hat{p}_n$ で表現できるとする．このとき，隠蔽定数 $s_{\hat{p}_i}(i = 1, \ldots, n)$ を演算 $\otimes$ を用いて結合し，この場合を項 $s_{\hat{p}_1} \otimes \ldots \otimes s_{\hat{p}_n}$ で表す．

$\otimes$ は，隠蔽定数が表す状態空間の共通部分をとる演算である．より明確に議論するために，演算 set の存在を仮定する．set は，ある条件を満たす隠蔽定数を引数に取り，それが表現している状態の集合を返す演算とする．この集合は，次のように各 $s_{\hat{p}_i}$ が表す状態

の集合の共通部分として表すこともできる:

$$set(s_{\hat{p}_1} \otimes \ldots \otimes s_{\hat{p}_n}) = set(s_{\hat{p}_1}) \cap \ldots \cap set(s_{\hat{p}_n})$$

本論文では，$s_{\hat{p}_1} \otimes \ldots \otimes s_{\hat{p}_n}$ のような項を合成隠蔽定数と呼ぶ．以下では，合成隠蔽定数について，より厳密な定義を与える．最初に，$\otimes$ による結合の最小単位である原始隠蔽定数を，以下のように定義する:

定義 4.1. 原子隠蔽定数 (*atomic state constant*)
前述した条件を満たす述語 $\hat{p}$ が成立する任意の状態を表す隠蔽定数を，原始隠蔽定数と呼ぶ．

定義 4.2. 合成隠蔽定数 (*composite state constant*)
合成隠蔽定数は，次の規則から作られるものに限る:

- 原子隠蔽定数は合成隠蔽定数である．
- 2 つの合成隠蔽定数 s_1 と s_2 を，演算 $\otimes$ で結んだ $s_1 \otimes s_2$ は，合成隠蔽定数である．

演算 $\otimes$ は，任意の原子隠蔽定数の組み合わせに対して合成隠蔽定数を返す．$\otimes$ は前述したように共通部分と同義の演算であるので，空集合であるような合成隠蔽定数も考えられる．例えば，$\hat{p}$ と $\neg\hat{p}$ が同時に成立することはないため，$set(s_{\hat{p}} \otimes s_{\neg\hat{p}})$ は空集合である．このような項は，ありえない場合を表している．

4.1.2 合成隠蔽定数を用いた検証

ここでは，合成隠蔽定数を用いて，振舞遷移機械のある性質 $prop : \Upsilon \to \{true, false\}$ を検証する手続きを考える．この手続きを概観すると，次の通りである．

- 場合分けに用いる各述語に着目し，その述語が成立する状態を表す，原子隠蔽定数を宣言する．
- 個々の場合を合成隠蔽定数で表現し，*prop* が成立することを確かめる．

例として，$\hat{p}$ と $\hat{q}$ に着目した場合分けが必要である検証を取り上げる．原子隠蔽定数として，$s_{\hat{p}}$，$s_{\neg\hat{p}}$，$s_{\hat{q}}$，$s_{\neg\hat{q}}$ を用意する．考慮すべき場合を表した合成隠蔽定数は，$s_{\hat{p}} \otimes s_{\hat{q}}$，$s_{\neg\hat{p}} \otimes s_{\hat{q}}$，$s_{\hat{p}} \otimes s_{\neg\hat{q}}$，$s_{\neg\hat{p}} \otimes s_{\neg\hat{q}}$ の 4 つである．最後に，これらの合成隠蔽定数で表された状態において，*prop* が成立することを簡約によって確かめればよい．

しかし，合成隠蔽定数が空集合を表している場合については問題が残る．ある合成隠蔽定数が空集合を表すとき，それは，そのような状態は存在しないことを意味している．こ

のため，$prop$ が成立するかどうかを議論するのは無意味である．従って，合成隠蔽定数が空集合を表すかどうかを検査し，空集合ならば，その場合は無視できるとよい．つまり，各場合を表した合成隠蔽定数 s について，各々，次の推論を行えばよい:

$$(set(s) \neq \emptyset) \Rightarrow prop(s)$$

含意 $(\Rightarrow)$ の性質を利用し，合成隠蔽定数 s が空集合でないことを前提とし，$prop(s)$ を議論する．s が空集合だったときは，s で表される場合は $prop$ が成立するかどうかの議論に影響を与えないという意味で，簡約結果として真が得られる．

任意の s について，$set(s) \neq \emptyset$ の完全な計算を実現することは，モデルが表す意味に関する議論が必要になり，困難である．例えば，列車が区間 t に存在することを表す述語を p，区間 t の列車数が 1 以上であることを表す述語を q とする．このとき，区間 t に列車が存在するにもかかわらず，区間 t には列車が 0 である場合を表す $s_p \otimes s_{\neg q}$ は，空集合を表している．しかし，このような判定を一般的に行わせる手続きを実現することは，困難である．

そこで，本論文では，健全な $set(s) \neq \emptyset$ の実現法について述べる．健全であるとは，$set(s) \neq \emptyset$ が偽を返すとき，s は空集合である．

健全な $set(s) \neq \emptyset$ は，単純な探索によって実現できる．今，原始隠蔽定数 $s_1 \otimes \ldots \otimes s_n$ (ただし，$n > 0$) が宣言されている．この時，各 s_i $(i = 1, \ldots n)$ について，s_i と結合すると空集合となる原子隠蔽定数の集合 $tbl(s_i)$ を与える．健全な $set(s) \neq \emptyset$ は，s に含まれる各原子隠蔽定数 s' について，s と $tbl(s')$ を比較し，共通する原子隠蔽定数の有無を判定する．もし一つでも共通する原子隠蔽定数が見つかったならば，s は空集合である．

4.1.3　CafeOBJ における合成隠蔽定数の実現

ここでは，前述した合成隠蔽定数を CafeOBJ で実現する方法について述べる．

振舞遷移機械 $\mathcal{S}$ の状態空間 Υ を表現している隠蔽ソート H に対し，合成隠蔽定数を表すソートを C，原子隠蔽定数を表すソートを A とし，A を C の C を H のサブソートとする宣言を行う:

```
*[ A < C < H ]*
```

演算子 $\otimes$ は，結合律と交換律が成り立つ中置演算子 `o` として，次のように宣言する:

```
op _o_ : C C -> C { assoc comm }
```

次に，原子隠蔽定数を宣言する．相補対 $\hat{p}$ と $\neg\hat{p}$ が成り立つ任意の状態を表すものとして，次の 2 つの原子隠蔽定数をソート A 上に宣言する:

```
ops s_p̂ s_¬p̂ : -> A
```

合成隠蔽定数の中に $s_{\hat{p}}$ が表れるとき，述語 $\hat{p}$ が真であることは，次の等式で表現できる．$s_{\neg\hat{p}}$ についても同様の等式を与える:

ceq $\hat{p}$($s_{\hat{p}}$ o c) = true if comp($s_{\hat{p}}$, c) .

ceq $\hat{p}$($s_{\neg\hat{p}}$ o c) = false if comp($s_{\neg\hat{p}}$, c) .

等式中の c は，ソート C 上の CafeOBJ 変数である．等式の条件部分に表れる演算 comp は，前述した健全な $set(s) \neq \emptyset$ の引数部分を効率のために変更し，第一引数には原子隠蔽定数，第二引数には合成隠蔽定数を渡すように限定したものである．

comp は，次のように定義する:

op comp : A C -> Bool

eq comp(a, c) = ¬(isin(c,tbl(a))) .

等式中の a は，ソート A 上の CafeOBJ 変数である．演算 tbl は，前述した tbl の CafeOBJ 上の表現である．isin は，c に含まれる原子隠蔽定数が，tbl が返す集合に存在するとき真を，そうでないときには偽を返す述語である．

合成隠蔽定数を用いた簡約は，次のように行われる．例えば，$s_{\hat{p}}$ を含む合成隠蔽定数 $\ldots \otimes s_{\hat{p}} \otimes \ldots$ を考える．$\ldots \otimes s_{\hat{p}} \otimes \ldots$ で表される任意の状態における $\hat{p}$ の値を得るためには，CafeOBJ の 'red' コマンドを用いて，次の指示を与える:

red $\hat{p}$(... o $s_{\hat{p}}$ o ... o s) .

演算 o には結合律と交換律が成立するため $s_{\hat{p}}$ の出現位置に関係なく，この項は前述の等式の左辺にマッチする．$\ldots \otimes s_{\hat{p}} \otimes \ldots$ に，$s_{\hat{p}}$ と結合すると空集合になるような原子隠蔽定数が含まれているならば，等式の条件が成立しないので，これ以上の書き換えは起こらない．そうでないならば，true に書き換えることができるので，$\hat{p}$ の値として true が得られる．

以上の記述を用いることで，述語に着目した場合分けの各場合を，項で表現できる．

4.2 安全性検証の組織化

4.2.1 基本の場合分け

帰納法で証明したい性質を $prop : \Upsilon \rightarrow \{true, false\}$ で表す．帰納法は，基底と帰納段階の 2 つの手続きに分けられる．基底については，初期状態を表す各隠蔽定数 s において $prop(s)$ が成立することを確かめればよく，検証に際して工夫の余地は少ない．ここでは，特に帰納段階について取り上げる．帰納法の仮定に由来する場合分けを一方の軸，振舞遷移機械の各遷移規則の効力条件に由来する場合分けをもう一方の軸に配し，場合分けをマトリクス状に整理する手法を提案する．この手法を基本の場合分けと呼ぶ．振舞遷移機械

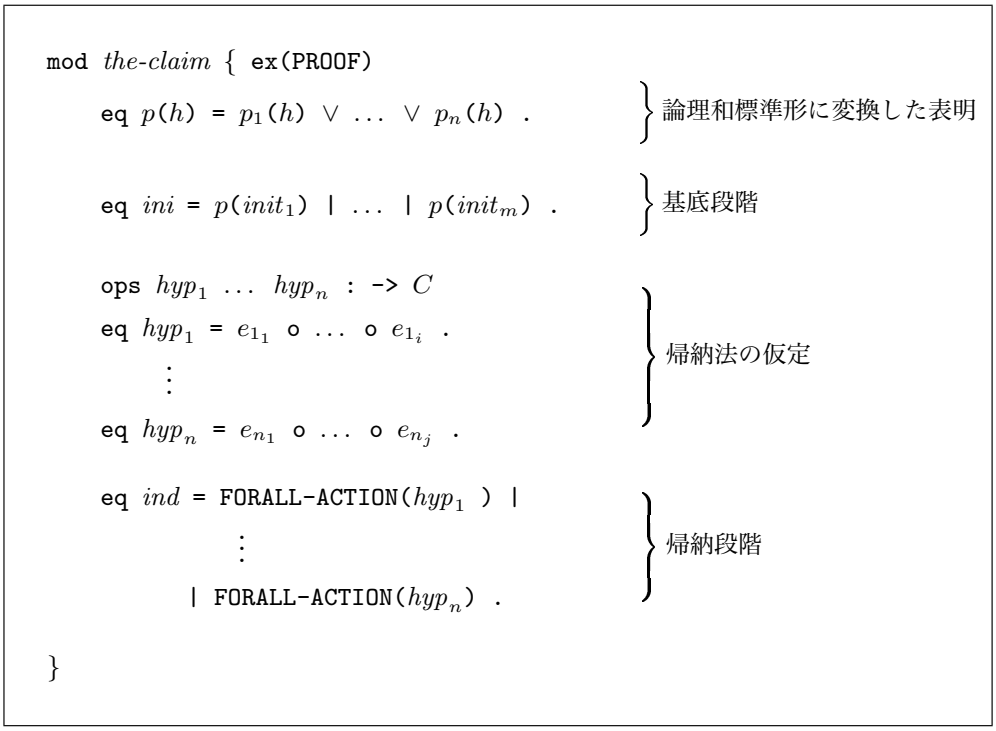


図 4.1: 安全性検証のひな型

が与えられると各遷移規則の効力条件は決まるが，一つの振舞遷移機械に対して示したい性質は複数考えられる．そこで，帰納法の仮定に由来する場合分けを配した軸をパラメタとして，示したい性質毎にマトリクスを再利用できるようにした．

帰納段階では，*prop* が成立する任意の状態 $s \in \Upsilon$ について，各遷移規則 $\tau \in \mathcal{T}$ が *prop* を保存することを示す．つまり，各 τ について，次の式で表される推論を行う:

$$\forall s.(prop(s) \Rightarrow prop(\tau(s)))$$

帰納法の仮定に由来する場合分けでは，大きく，$prop(s)$ が真である場合と偽である場合について，場合分けが必要である．しかし，含意の性質から，$prop(s)$ が偽である場合については，上記の式が真であることは自明である．よって，$prop(s)$ が真である場合についてのみ議論を行えば十分である．

ここで，ある τ について検証する手続きについて考える．このとき，大きく効力条件 $c_\tau(s)$ が真である場合と偽である場合について，場合分けが必要である．しかし，仕様が正しく記述されているならば，$c_\tau(s)$ が偽の場合は，振舞遷移機械の定義から，τ はいかなる観測の値も変えないので，τ が *prop* を保存することは自明である．よって，$c_\tau(s)$ が真である場合についてのみ議論を行えば十分である．

この考察に基づき，帰納段階における基本的な場合分けを，次のように与える．最初

に，c_τ と $prop$ を論理和標準形に変換する:

$$c_\tau(s) = c_{\tau_1}(s) \vee \ldots \vee c_{\tau_m}(s)$$

$$prop(s) = prop_1(s) \vee \ldots \vee prop_n(s)$$

これらから，c_τ と $prop$ が共に真である場合を網羅的に議論するには，以下の行列に示す $n \times m$ 通りの場合分けを行い，各条件 $c_{\tau_i}(s) \wedge prop_j(s)$ $(0 \leq i \leq m, 0 \leq j \leq n)$ が成立する各状態 s について $prop(s) \Rightarrow prop(\tau(s))$ が真であることを確める:

$$\begin{array}{ccc} c_{\tau_1}(s) \wedge prop_1(s) & \ldots & c_{\tau_1}(s) \wedge prop_n(s) \\ \vdots & \ddots & \vdots \\ c_{\tau_m}(s) \wedge prop_1(s) & \ldots & c_{\tau_m}(s) \wedge prop_n(s) \end{array}$$

次に，基本の場合分けを合成隠蔽定数で表現する手法について述べる．論理和標準形の各節 $c_{\tau_i}(s)$ や $prop_j(s)$ は，次の形をしている:

$$l_1(s) \wedge \ldots \wedge l_k(s)$$

l_r (r = 1, ..., k) は，リテラルである．これらに対応する原子隠蔽定数 s_r を，それぞれ宣言する．すると，各節 $l_1 \wedge \ldots \wedge l_k$ は，合成隠蔽定数 $s_{l_1} \otimes \ldots \otimes s_{l_k}$ で表現できる．

これまでの議論から，合成隠蔽定数を用いた帰納法の帰納段階は，次のようになる．$c_{\tau_i}(s)$ が成立する状態を合成隠蔽定数 $s_{c_{\tau_i}}$ で，$prop_j(s)$ が成り立つ状態を s_{prop_j} で表す．これらを用いて，$c_{\tau_i}(s) \wedge prop_j(s)$ を満たす状態において，操作演算 τ が $prop$ を保存することは，以下の式で記述できる:

$$set(s_{c_{\tau_i}} \otimes s_{prop_j}) \neq \emptyset \Rightarrow (prop(s_{c_{\tau_i}} \otimes s_{prop_j}) \Rightarrow prop(\tau(s_{c_{\tau_i}} \otimes s_{prop_j})))$$

論理和標準形に変換した $c_\tau(s)$ や $prop(s)$ の各節を合成隠蔽定数で表現し，これらの組み合わせについて，上に示した式を表現する項の簡約を繰り返すことで，τ が $prop$ を保存することを検証できる．幸運にもすべての組み合わせについて `true` が得られたならば，検証は成功している．ある組み合わせについて，`true` が得られなかったならば，その組み合わせが示す場合において，検証が成功しなかったを示している．その場合が真に反例を示しているのか，到達不能な場合であるのか，さらに精緻な場合分けが必要であるのか，という判断は検証者に委ねられる．

4.2.2 安全性検証の表現

基本の場合分けは，検証したい性質毎に再利用可能なマトリクスを提供する．以下では，CafeOBJ コードのレベルで，どのように再利用が行われているのかを示す．

基本の場合分けを用いて，安全性検証を行うためのひな型を図 4.1に示す．検証したい表明 p ごとに CafeOBJ のモジュール *the-claim* を以下のように作成する．

1. 表明 p の論理和標準形 $p_1 \vee \ldots \vee p_n$ を，等式で表現する．
2. 基底の推論を演算 *ini* に指定する．つまり，仕様に現れる初期状態を表す各隠蔽定数 $init_k (k = 1, \ldots, m)$ について，p が成立することを調べる．
3. 帰納法の仮定を合成隠蔽定数で表現する．つまり，表明の各節 p_k $(k = 1, \ldots, n)$ について，合成隠蔽定数 hyp_k を宣言し，p_k を合成隠蔽定数で表現した $e_1 \otimes \ldots \otimes e_v$ を等式で与える．
4. 帰納段階の推論を演算 *ind* に指定する．つまり合成隠蔽定数 hyp_k $(k = 1, \ldots, n)$ で表された各場合について，すべての操作演算が p を保存することを調べる．

検証を行うには，処理系を用いて項 *ini* と *ind* を簡約すればよい．

演算 `|` は推論を表す式を連結しているだけである．ただし，`|` には結合律，交換律およびべき等律が成り立つものとする．`|` で連結されたすべての推論が，`true` まで簡約されたとき，検証が完了したことがわかる．`|` にべき等律が宣言されているため，`true | true` は `true` に簡約され，検証が成功したときは，単に `true` だけの項になる．

一方で，複数の推論を一つの項で表しているため，単純に連結しただけでは，`true` まで簡約されずに残った項が，どの場合に関する推論であるかが判然としない．これを解決するため，各場合に関する推論を表す演算を宣言する．各遷移規則 $\tau_{j_1,\ldots,j_m}$ について，ある場合について推論するための項 $case_\tau$ を以下のように宣言する:

```
op case_τ : V_j1 ... V_jm C C -> Bool
ceq case_τ(v_j1, ..., v_jm, c, c') = true
  if comps(c o c') ⇒ (prop(c o c')
    ⇒ prop(τ(j1, ..., jm, c o c')))
```

等式中，$v_{j_1}, \ldots, v_{j_m}$ は可視ソート $V_{j_1}, \ldots, V_{j_m}$ の，c と c' は隠蔽ソート C 上の CafeOBJ 変数である．$case_\tau$ は，$c \otimes c'$ で与えられる場合について，$\tau_{j_1,\ldots,j_m}$ が *prop* を保存するかどうかを検査する．アリティに現れる C には，それぞれ $\tau_{j_1,\ldots,j_m}$ の効力条件の各節 $c_{\tau_{j_1,\ldots,j_m}}$ を表現する合成隠蔽定数 $s_{e_{\tau_i}}$ と，帰納法の仮定 *prop* の各節を表現する s_{prop_j} を与える．$case_\tau$ では，実際の推論を行う項が，等式の条件部分に記述されているので，最初に条件部分の簡約が行われ，真が得られたとき，その $case_\tau$ は `true` に書き換えられる．そうでなかったときは，$case_\tau$ が書き換えられずに残るため，どの場合において，どの操作演算を適用したとき，`true` まで簡約できなかったかが判明する．

演算 `FORALL-ACTION` は，前述したマトリクスの列を表現する項である．この演算は，

以下のように宣言する:

```
op FORALL-ACTION : C -> Bool
eq FORALL-ACTION(c:C)
   = case_τ1(c, s_cτ1)
   | case_τ2(c, s_cτ2)
   | ... .
```

FORALL-ACTION に，帰納法の仮定を表している合成隠蔽定数 $s_{prop_1}, \ldots, s_{prop_m}$ を与えることで，基本の場合分けを伴った帰納段階の検証を行うことができる．

4.2.3 スタフ閉そくの安全性検証

ここでは，前述した手法を用いて，スタフ閉そくの安全性を検証する．スタフ閉そくの安全性は，次の表明で表される．

表明 4.1　任意の到達可能な状態 s において，任意の列車 n と m(ただし $n \neq m$) が同時に区間 t4 に進入しない．

$$\neg(pos_n(s) = \mathsf{t4} \wedge pos_m(s) = \mathsf{t4})$$

しかし，本論文では，より強い表明を検証することによって，表明 4.1 を示す．実際に検証する表明を以下に示す．

表明 4.2　任意の到達可能な状態 s において，任意の列車 n と m (ただし $n \neq m$) が区間 t3，t4 および t5 を合わせた区間に同時に進入しない．

$$\begin{aligned}&\neg((pos_n(s) = \mathsf{t3} \vee pos_n(s) = \mathsf{t4} \vee pos_n(s) = \mathsf{t5})\\ \wedge\ &(pos_m(s) = \mathsf{t3} \vee pos_m(s) = \mathsf{t4} \vee pos_m(s) = \mathsf{t5}))\end{aligned}$$

原始隠蔽定数の宣言

モジュール CASES において，検証に使用する原始隠蔽定数を宣言した．

最初に，演算 o などを，前述した定義に従って記述した．

次に，表明に表れる任意の列車 n と m に対応する定数として，c1 と c2 を宣言した．

```
op c1 c2 :  -> TrID
eq (c1 = c2) = false .
eq (c1 = a) = false .
eq (c1 = b) = false .
```

```
eq (c2 = a) = false .
eq (c2 = b) = false .
```

駅の識別子 a や b との等価関係を偽とする等式は，可視ソート StaffPos 上の等価関係を宣言している．列車 c1 や c2 と駅 a や b は，明らかに等しくないので，偽とした．

次に，各列車について，位置を表す原始隠蔽定数を宣言した．仕様では，ある列車 c が，区間 tl に存在することを判定する述語は，次のように記述されている．

```
pos(c:TrID, s:State) = tl
```

従って，任意の列車 c1 が，区間 tl にいることを表す原始隠蔽定数 c1tl と，そうでないことを表す ~c1tl を定義した．

```
ops c1tl ~c1tl :  -> A
```

c1tl の意味は，次のように定義できる:

```
ceq (pos(c1, ( c1tl o c:C)) = tl) = true if comp( c1tl, c) .
```

しかし，この定義は次の理由から不便である．例えば，列車 c が tl に存在する場合を議論しているとき，仕様に現れる他の述語 pos(c, s) = tr の簡約を考える．列車 c は tl にいることが分かっているので，この述語は偽であるとして簡約したい．TcID 上の残りの各要素についても，同様のことが言える．しかし，上のような定義であると，tr のみならず TcID の残りの各要素についても，すべて以下のような等式を宣言しなければならない．

```
ceq (pos(c1,( c1tl o c:C)) = t1) = false if comp( c1tl, c) .
    ⋮
ceq (pos(c1,( c1tl o c:C)) = tr) = false if comp( c1tl, c) .
```

そこで，一括してこのような等式を記述するため，宣言を工夫し，次の定義とした:

```
ceq (pos(c1,( c1tl o c:C)) = t:TcID) = (t = tl) if comp( c1tl, c) .
ceq (pos(c1,(~c1tl o c:C)) = tl) = false  if comp(~c1tl, c) .
```

残りの区間 t1, ..., tr についても，同様の原子隠蔽定数 c1t1, ..., c1tr を宣言した．任意の列車 c2 についても，同様に宣言した．なお，以下では，~ が前置されている原子隠蔽定数は，その述語が偽である状態の代表元を表している．

次に，これらの原子隠蔽定数と，結合すると空集合となる原子隠蔽定数の集合を与える．ある状態 s において，列車 c1 が tl に存在するならば，c1 は他の区間に存在し得ない．従って，c1tl と結合した際に空集合となるのは，同じ述語が偽となる場合を表す ~c1tl と，c1 が他の区間に存在する場合を表す各原子隠蔽定数 c1t1，...，c1t7 および c1tr である．従って，tbl(c1tl) は次のように記述した:

```
eq tbl( c1tl) = ~c1tl ::  c1t1 ::  c1t2 ::  c1t3
```

```
          ::  c1t4 ::  c1t5 ::  c1t6 ::  c1t7 ::  c1tr .
```

逆に，列車 c1 が tl に存在しない場合を考えるとき，c1 が tl に存在する場合だけが否定できる．従って，~c1tl と結合すると空集合となると判断できるのは，c1tl ただ一つである．従って，tbl(~c1tl) は次のように記述した:

```
eq tbl(~c1tl) = c1tl .
```

同様に，列車が方向 r や l に走っている場合を表す原始隠蔽定数 c1r，c1l，c2r，c2l や，スタフが列車または駅にある場合を表す sc1，sc2，sa，sb，および，これらを否定する原子隠蔽定数を，同様に宣言した．

帰納法の検証を表現する項

基本の場合分けを行う帰納法の検証を表現する項に関する宣言を，モジュール PROOF に記述した．

最初に，演算 | などを，前述した定義に従って記述した．

次に，操作演算 move の個々の場合について検証する項 MOVE を，前述した定義に従って，次のように記述した．

```
op MOVE : TrID TcID TrDir C C -> Bool
ceq MOVE(TR:TrID, TC:TcID, TD:TrDir, C:C, C':C) = true
    if comps(C, C') ⇒
        (p(C o C') ⇒ p(move(TR, TC, TD, (C o C')))) .
```

操作演算 catch と release についても，同様に CATCH と RELEASE を記述した．

項 FORALL-ACTION に，操作演算と，その操作演算の効力条件を合成隠蔽定数で表したものを与えた．

```
eq FORALL-ACTION(C:Case) =
      MOVE(c1, tl, r, c1tl o c1r, C)
    | MOVE(c1, t1, r, c1t1 o c1r o sc1, C)
        ⋮
```

表明 4.2 の検証

証明　モジュール PROOF を輸入し，モジュール CLAIM4-2 を定義した．

最初に，表明 4.2 を論理和標準形で表現する．これは，次のように記述できる．

```
ceq p(S:State) =
```

```
    ((¬ pos(c1, S) = t3) ∧ (¬ pos(c1, S) = t4)
                          ∧ (¬ pos(c1, S) = t5))
  ∨ ((¬ pos(c2, S) = t3) ∧ (¬ pos(c2, S) = t4)
                          ∧ (¬ pos(c2, S) = t5))
  if ¬ (c1 = c2) .
```

次に，基底段階に関して記述を行う．初期状態 init-a と init-b が，ともに p を満たすことを調べる．

```
eq INI = p(init-a) | p(init-b) .
```

帰納段階についての記述を行う．最初に，帰納法の仮定を表す合成隠蔽定数を記述する．論理和標準形に変換した表明 4.2 は，2 つの節から構成されている．そこで，それらの節を表す項 hyp1 と hyp2 を宣言し，各節を合成隠蔽定数で表したものを等式で与えた．

```
ops hyp1 hyp2 :  -> Case
eq hyp1 = ~c1t3 o ~c1t4 o ~c1t5 .
eq hyp2 = ~c2t3 o ~c2t4 o ~c2t5 .
```

帰納段階では，FORALL-ACTION に，hyp1 と hyp2 を渡し，帰納段階について，基本の場合分けを伴う検証を行うこととした．

```
eq IND = FORALL-ACTION (hyp1)
       | FORALL-ACTION (hyp2) .
```

これらは，CafeOBJ のコマンドラインから，次のコマンドを与えることで，簡約することができる．

```
open CLAIM4-2
red INI | IND .
close
```

以下に，簡約結果を示す．

```
-- reduce in CLAIM4-1 :  INI | IND
MOVE(c1,t1,r,c1t1 o c1r o sc1, ~c1t3 o ~c1t4 o ~c1t5)
| MOVE(c1,t7,l,c1t7 o c1l o sc1, ~c1t3 o ~c1t4 o ~c1t5)
| MOVE(c2,t1,r,c2t1 o c2r o sc2, ~c2t3 o ~c2t4 o ~c2t5)
| MOVE(c2,t7,l,c2t7 o c2l o sc2, ~c2t3 o ~c2t4 o ~c2t5)
| true :  Bool
(0.000 sec for parse, 12557 rewrites(23.260 sec), 108537 matches)
```

この結果は，| で結ばれた 4 つの場合が true まで簡約できなかったことを表している．結果の最初の項:

MOVE(c1,t1,r,c1t1 o c1r o sc1, ~c1t3 o ~c1t4 o ~c1t5)

は，合成隠蔽定数 c1t1 o c1r o sc1 o ~c1t3 o ~c1t4 o ~c1t5 の場合において，操作演算 move(c1,t1,r,s) が失敗したことを表している．c1t1 o c1r o sc1 は，操作演算の効力条件，~c1t3 o ~c1t4 o ~c1t5 は hyp1 に由来している．これらの合成隠蔽定数は，次のことを示唆している．スタフは列車 c1 が持っており (sc1)，それゆえ c1 は t3 に移動している．一方，列車 c2 の位置については，何も議論していないので，もし列車 c2 が t3，t4 または t5 にいるならば，2 つの列車が t3，t4 または t5 に同時に進入し，表明は崩れる．しかし，スタフ閉そくのモデルから，列車がそれらの区間に入るためには，スタフを所有していなければならず，もしある列車が t3，t4 または t5 にいるならば，スタフを所有しているはずである．従って，列車の位置とスタフの関係ついて，補題が必要である．

補題 4.1 の検証

補題 4.1　任意の到達可能な状態 s において，任意の列車 n が区間 t3，t4 または t5 に存在するとき，その列車がスタフを所有している:

$$(pos_n(s) = \mathsf{t3} \vee pos_n(s) = \mathsf{t4} \vee pos_n(s) = \mathsf{t5}) \Rightarrow staff(s) = n.$$

証明　モジュール PROOF を輸入し，モジュール LEMMA4-1 を定義した．

最初に，補題 4.1 を論理和標準形で表現する．これは，次のように記述した．

```
eq p(S:State) = (¬ pos(c1, S) = t3) ∧ (¬ pos(c1, S) = t4)
                ∧ (¬ pos(c1, S) = t5)
    ∨ (staff (S) = c1) .
```

表明 4.2 と同様に，補題 4.1 についても基底段階と帰納段階について記述した．補題 4.1 を検証する項 INI | IND の簡約結果を以下に示す．

```
-- reduce in LEMMA4-1 :  INI | IND
true :  Bool
(0.000 sec for parse, 4391 rewrites(12.050 sec), 33012 matches)
```

補題 4.1 は，基本の場合分けを用いるだけで，検証することができた．□

補題を導入した表明 4.2 の検証

証明　補題 4.1 が証明できたので，これを表明 4.2 に導入する．帰納段階において，補題を導入する必要があるので，前述したモジュール CLAIM4-2 の中の，帰納段階の部分を書

き換える．

補題 4.1 は，ある列車 c が区間 t3，t4 または t5 にいないか，その列車がスタフを所有しているか，これらのどちらかが成り立っていることを意味している．従って，hyp1 と hyp2 に関する議論を，それぞれ 2 つに分けて，補題を導入した．

```
eq IND = FORALL-ACTION(hyp1 o sc2 o s)
    | FORALL-ACTION(hyp1 o ~c2t3 o ~c2t4 o ~c2t5 o s)
    | FORALL-ACTION(hyp2 o sc1 o s)
    | FORALL-ACTION(hyp2 o ~c1t3 o ~c1t4 o ~c1t5 o s) .
```

簡約結果を以下に示す．

```
-- reduce in CLAIM4-2 :  INI | IND
true :  Bool
(0.000 sec for parse, 18429 rewrites(26.790 sec), 131276 matches)
```

表明 4.2 は，補題 4.1 を用いることで，検証することができた．□

4.3　場合分けの自動生成

前節では，合成隠蔽定数を用い，基本の場合分けを用いて，帰納段階の検証を行う手法を述べた．この手法では，表明 4.2 の例で示したように，true まで簡約できない場合が検証者に示され，検証者はそれを分析することで補題を推測できた．ここでは，true に至らなかった場合を対象とし，より詳細な場合分けを自動生成し，それらの場合について検証を試みる手法を提案する．この手法は，検証者に検証に有効な場合分けや補題を発見するための手掛かりを提供する．

項書換えを用いた検証において，場合分けが完了するとは，条件式に現れる原始述語の真偽が決定できるということである．これらの真偽がすべて決定できるならば，表明は真または偽まで簡約できる．真まで簡約できたならば，検証の成功を示し，偽に簡約できたならば，反例か，その場合が到達不能であることを示している．例えば，表明 4.2 の例では，列車 c2 に関する位置情報がなかったため，述語 pos(c2, s) = t3 などの真偽値が決定できず，表明を真にも偽にも簡約できない場合が残っていた．

ここでは，以下に示す手法を提案する．合成隠蔽定数で表される場合 c について，ある操作演算 act を適用したときに関する推論が，真にも偽にも簡約できなかったとする．このとき，原始隠蔽定数の相補対の集合 $\{(e_{p_i}, e_{\neg p_i})\}$ $(i = 1, \ldots, n)$ を用い，各 i について，場合 $c \otimes e_{p_i}$ と $c \otimes e_{\neg p_i}$ を作り，項 $p(act(c \otimes e_{p_i}))$ と $p(act(c \otimes e_{\neg p_i}))$ を簡約する．各 i に対する簡約結果は，次のことを示している:

簡約結果が共に真　場合 $c \otimes e_{p_i}$ と $c \otimes e_{\neg p_i}$ を用いた場合分けを追加することで，検証が成功する．

(どちらか、または両方の) 簡約結果が偽　場合 $c \otimes e_{p_i}$ と $c \otimes e_{\neg p_i}$ を用いることで，充分な場合分けを行うことができる．これらのうち，偽になった場合は反例または到達不能な状態を表している．

(どちらか、または両方の) 簡約結果が真でも偽でもない　場合 $c \otimes e_{p_i}$ と $c \otimes e_{\neg p_i}$ では，場合分けが不足している．

なお，簡約結果の判定は，上記の記述順で行うものとする．検証者はこの結果に基づき，検証を進めることができる．

上記の結果のうち，簡約結果が偽になり，検証者がその場合を到達不能であると判断したものについては，補題を推測する必要がある．これについて本論文では次の 2 つの戦略を提案する:

- 原始隠蔽定数 e と合成隠蔽定数 c で表される場合 $e \otimes c$ において，操作演算 act の効力条件が真にならないことを補題とする．つまり，効力条件が成立する場合を表す合成隠蔽定数 c' と結合した場合 $e \otimes c \otimes c'$ が，到達不能であることを証明する．
- 原始隠蔽定数 e と合成隠蔽定数 c で表される場合 $e \otimes c$ において，$e \otimes c$ が空集合となることを補題とする．つまり，$e \otimes c$ が到達不能であることを証明する．

4.3.1 CafeOBJ による実現

ここでは，前述した探索を CafeOBJ で実現する方法について述べる．

演算 `traverse` は，原始隠蔽定数の相補対の集合と合成隠蔽定数で表された場合を引数にとり，判定結果を返す．相補対は，演算`<< _ , _ >>`で表し，その集合はリストで表現した．このリストは，ソート `PairList` 上に定義した．演算 `empty` は空リストを表し，`_::_`はリストの構成子である．

ある場合 c と，相補対のリストに列挙されている各原子隠蔽定数からなる場合について，検証を試みる演算 `traverse` は図 4.2のように記述した．演算 `try` は，1 つの原始隠蔽定数 a と，1 つの合成隠蔽定数 c をとり，簡約結果を表す原始隠蔽定数を演算である．`try` は場合 $a \otimes c$ において，操作演算 `act` を適用したときの簡約結果を返す．簡約結果が真のときは，特別な原子隠蔽定数 `tt` を，偽のときは `ff` を，それ以外のときは a をそのまま返す．なお，`act` は仮引数であり，実際に解析を行う際には，特定の操作演算を指定する．`if _ then _ else _ fi` は，CafeOBJ 組込みの演算であり，関数型言語の if 文と

```
eq  traverse (empty, C:Case) = empty .
eq  traverse (<< A1:Atom, A2:Atom >> :: L:PairList, C:Case)
    = << try(A1, C), try(A2, C) >> :: traverse(L, C) .

bop act : State -> State
eq  try (A:Atom, C:Case)
    = if (comp(A, C) => (p(A o C) => p(act(A o C)))) == true
         -- 簡約結果が true になった
         then tt
         else if (comp(A, C) => (p(A o C) => p(act(A o C)))) == false
                 -- 簡約結果が false になった
                 then ff
                 -- それ以外
                 else A
              fi
      fi .
```

図 4.2: 場合分けを自動生成するための CafeOBJ コード

```
open CLAIM4-2
eq act(S:State) = move(c1, t1, r, S) .
red traverse(pos-c2, c1t1 o c1r o sc1 o ~c1t3 o ~c1t4 o ~c1t5) .
close
```

図 4.3: 場合分けを自動生成するための証明譜

同様の定義である.

4.3.2　スタフ閉そくへの適用

スタフ閉そくの表明 4.2 について，本手法を適用した.

表明 4.2 では，次の 4 つの場合について，true が得られなかった.

- MOVE(c1,t1,r,c1t1 o c1r o sc1, ~c1t3 o ~c1t4 o ~c1t5)
- MOVE(c1,t7,l,c1t7 o c1l o sc1, ~c1t3 o ~c1t4 o ~c1t5)
- MOVE(c2,t1,r,c2t1 o c2r o sc2, ~c2t3 o ~c2t4 o ~c2t5)
- MOVE(c2,t7,l,c2t7 o c2l o sc2, ~c2t3 o ~c2t4 o ~c2t5)

ここでは，最初の場合についてのみ示す. 場合 c1t1 o c1r o sc1 o ~c1t3 o ~c1t4 o ~c1t5 において，move(c1,t1,r,s) を適用した時が問題となっている. 従って，仮引数 act に，move(c1,t1,r,s) を与えるために，図 4.3に示す証明譜を記述した. なお，pos-c2 は，列車 c2 の位置に関する原子隠蔽定数の相補対の集合を保持している.

```
-- opening module CLAIM4-2.. done.*
-- reduce in %CLAIM4-2 :
       traverse (pos-c2, c1t1 o c1r o sc1 o hyp1)
   << tt , ~c2tl >> :: << tt , ~c2t1 >> :: << tt , ~c2t2 >>
:: << ff , ~c2t3 >> :: << ff , ~c2t4 >> :: << ff , ~c2t5 >>
:: << tt , ~c2t6 >> :: << tt , ~c2t7 >> :: << tt , ~c2tr >>
:: empty
(0.000 sec for parse, 8946 rewrites(17.600 sec), 72066 matches)
```

図 4.4: 場合分けを自動生成した結果

この証明譜を CafeOBJ に読み込ませ，図 4.4に示す結果を得た．この結果は，列車 c2 が t3，t4 または t5 にいる場合を示している c2t3，c2t4 および c2t5 を加えた場合に関する推論が，偽まで簡約できたことを示している．これらの場合は，到達不能な状態であると判断し，move(c1,t1,r,s) の効力条件が真に成立しないことを示す補題を立てた．このことは，補題 4.1 と一致している．

項書換えによる推論は健全性しか保証できないので，偽が返った場合が本当に反例を示しているのか，それとも到達不能な状態を示しているのかは，検証者の判定に委ねられる．しかし，以上の手続きによって，検証者が考慮すべき場合を減らすことができた．そして，検証者はこれらの情報に基づき，補題を推測することができた．

第 5 章

複線区間の鉄道信号システム

本章では，複線区間の鉄道信号システム [2][3] を例題として取り上げ，3 章で述べた手法により仕様を記述し，4 章で述べた手法により安全性を検証する．本章で記述する例題では，効力条件や表明から得られる原始隠蔽定数だけでは，場合分けが不足することが示される．しかし，検証のフレームワーク外で，原始隠蔽定数を必要に応じて宣言したり，従来の証明譜による場合分けの手法を組み合わせることで，このような問題も解くことができることを示す．

なお、本章で取り上げる例題は，筆者が [36] で示したものに，大幅な改善を施したものである．

5.1 概要

我が国の鉄道において，複線区間は左側通行による一方通行である．従って，我が国の複線区間の鉄道信号は，一方通行の線路において，列車同士の追突を防ぐことを目的としている．ここでは，複線区間の鉄道から片側の線路だけを取り出し，一方通行の線路を持つ鉄道システムを議論の対象とする．これを一方通行の鉄道システムと呼ぶ．

図 5.1に一方通行の鉄道システムの概略を示す．図の左側は，車庫である．そこに m 個の列車 $c_1, c_2, \ldots, c_m$ がいる．線路は図の右方向に (無限に) 延びており，連続した区間 $\mathsf{t0}, \mathsf{t1}, \mathsf{t2}, \ldots$ に区切られている．t0 (車庫) を除く各区間には，その区間に列車が進入してよいかどうかを示す信号機と，その区間に列車がいるかどうかを検知するセンサーが取り付けられている．センサーと信号機の識別子は，それが設置されている区間の識別子を共用する．

各信号機 $\mathsf{t}n$ $(n = 1, 2, \ldots)$ は，センサー t0 を参照し，後続の列車に対して，その区間

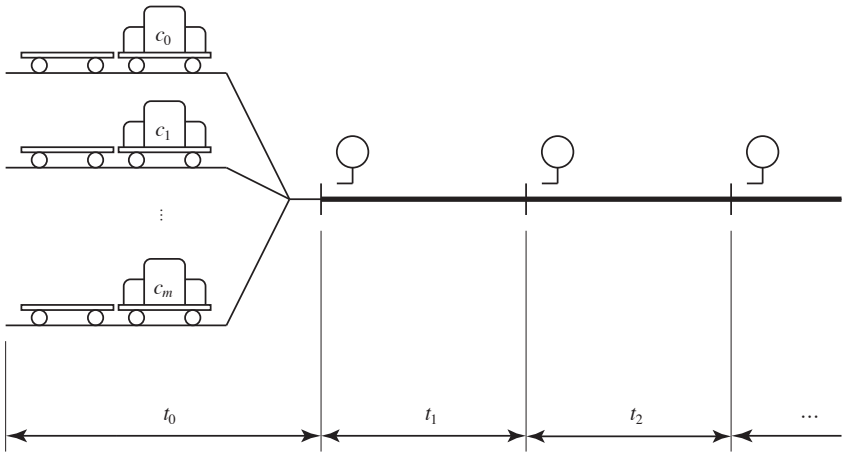


図 5.1: 一方通行の鉄道システム

に進入できるかどうかを知らせる．もし，センサーが列車を検知しているならば，信号は停止信号 (赤) を現示する．逆に，センサーが列車を検知していないならば，信号は進行信号 (青) を現示する．

すべての列車は，最初は区間 t0 にいて，区間単位で移動する．今，tn $(n = 1, 2, \ldots)$ にいる列車が，次の区間 t$(n+1)$ に進入しようとするとき，信号 t$(n+1)$ を参照し，それが進行信号であれば，区間 t$(n+1)$ に移動し，停止信号であれば，tn に留まる．

t0 (車庫) を除く各区間について，同時に 2 列車が進入することがなければ，一方通行の鉄道システムにおいて，列車同士の衝突は発生していないことが保証できる．

5.2 振舞遷移機械によるモデル化

一方通行の鉄道システムで使用するデータ型は，次の通りである．

- $C = \{0, 1, 2, \ldots\}$．列車の識別子．
- $T = \{\mathsf{t1}, \mathsf{t2}, \ldots\}$．(t0 を除く) 区間 (または，信号機およびセンサー) の識別子．
- $T' = \{\mathsf{t0}\} \cup T$．t0 を含む区間の識別子．次の区間を返す演算 $next : T' \to T$ が定義されている．
- $N = \{0, 1, 2, \ldots\}$．自然数の集合．

これらのデータを用いて，一方通行の鉄道システムを表す振舞遷移機械 $\mathcal{S} = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ を定義する．一方通行の鉄道システムを表す状態空間 Υ の存在を仮定する．

観測 $\mathcal{O}$ は次のように表現できる．

- $\{pos_c : \Upsilon \to T' \mid c \in C\}$．列車 $c \in C$ の現在位置を区間の識別子 T' で返す．

- $\{num_t : \Upsilon \to N \mid t \in T\}$．区間 t に入っている列車数を，自然数 N で返す．

なお，各信号機は演算 $green? : N \to \{\text{true}, \text{false}\}$．で表されている．$green?$ は，信号が進行信号の時に真を，停止信号の時に偽を返す．各区間 $t \in T$ の信号機は，$green?(num_t(u))$ で観測できる．

初期状態 $\mathcal{I}$ は，任意の状態 $u \in \Upsilon$ に対し，次のように表現できる．

- $\forall c \in C.pos_c(u) = \mathsf{t0}$．各列車は，t0 にいる．
- $\forall t \in T.num_t(u) = 0$．各区間 $t \in T$ には，列車がいない．

遷移規則 $\mathcal{T}$ は，各列車の移動を表す $move$ の集合である．遷移規則が適用された時の状態を $u \in \Upsilon$，遷移規則が適用された後の状態を $u' \in \Upsilon$ と表記する．

- $\{move_{(c,t)} : \Upsilon \to \Upsilon \mid c \in C, t \in T'\}$．列車 c が区間 t から，次の区間へ移動することを表す．効力条件は，$pos_c(u) = t \wedge green?(num_{next(t)})$ である．効果は、列車が次の区間で移動したことを表す $pos_c(u') = next(pos_c(u))$ と，列車が u においていた区間の列車数が 1 減少することを表す $num_t(u') = num_t(u) - 1$，およびその次の区間の列車数が 1 増加することを表す $num_{next(t)}(u') = num_{next(t)}(u) + 1$ である．

以上のように，一方通行の鉄道システムをモデル化した．

5.3　CafeOBJ による仕様記述

振舞遷移機械でモデル化した一方通行の鉄道システムを CafeOBJ で記述した．列車の識別子，区間の識別子および信号機を記述するモジュール TRAINID, TRACKID および SIGNAL と，一方通行の鉄道システム本体を記述するモジュール ONEWAY-RAILROAD-SYSTEM の 4 つのモジュールから成る．仕様は，全体で 75 行であり，小規模な例題に属すると考えられる．完全なコードは，付録 B.1 に示す．

5.3.1　抽象データ型の記述

モジュール TRAINID

列車の識別子 (集合 C) を定義するモジュールである．CafeOBJ 組込みのモジュール NAT を輸入し，ソート名を TrID に名前替えした．また，TrID 上の項の等価性を判定する述語 = を定義した．

モジュール TRACKID

区間の識別子 (集合 T) を定義するモジュールである．車庫を表す区間 t0 は特別な扱いとなっている．そこで，可視ソート mTcID と，そのスーパーソート TcID を宣言した．

演算 yard は，区間 t0 を表す定数である．演算 next は，次の区間を返す演算 $next$ を表している．従って，t1 や t2 は，next(yard) や next(next(yard)) と表すことができる．

また，TcID 上の項の等価性を判定する述語 = を定義した．

モジュール SIGNAL

信号とセンサーに関するデータ (集合 S) を定義するモジュールである．最初に，列車数を表すため，CafeOBJ 組込みのモジュール NAT を輸入した．次に，演算 $green?$ を宣言し，等式でその意味を与えた．

5.3.2 抽象機械の記述

一方通行の鉄道システムを表す抽象機械は，モジュール ONEWAY-RAILROAD-SYSTEM に記述した．最初に，5.3.1 節 で記述した各モジュールを輸入し，状態空間 Υ を表す隠蔽ソート State を宣言した．

観測 $\mathcal{O}$ は，次のように記述した．

```
bop pos :  TrID State -> TcID
bop num :  mTcID State -> Nat
```

初期条件 $\mathcal{I}$ は，次のように記述した．

```
op init :  -> State
eq pos(TR:TrID, init) = yard .
eq num(MTC:mTcID, init) = 0 .
```

遷移規則 $\mathcal{T}$ と，効力条件と効果を記述するための演算は，次のように記述した．

```
bop move :  TrID TcID State -> State
op c-move :  TrID TcID State -> Bool
op e-pos-move :  TrID TrID TcID State -> TcID
op e-num-move :  mTcID TrID TcID State -> Nat
```

move の効力条件は，次のように記述した．

```
eq c-move(TR:TrID, TC:TcID, S:State)
    = (pos(TR, S) = TC) ∧ green?(num(next(TC), S)) .
```

この等式は，$move_{(c,t)}$ の効力条件を，CafeOBJ でそのまま書き下したものである．

move の pos に対する効果は，次のように記述した．

```
ceq e-pos-move(TR:TrID, TR':TrID, TC':TcID, S:State)
        = next(pos(TR, S))
    if TR = TR' .
ceq e-pos-move(TR:TrID, TR':TrID, TC':TcID, S:State)
        = pos(TR, S)
    if not (TR = TR') .
```

1 つ目の等式は，move によって動いた列車 TR' の位置が，次の区間になることを意味している．2 つ目の等式は，TR' 以外の列車の位置は，変化しないことを意味している．

move の num に対する効果は，次のように記述した．

```
ceq e-num-move (MTC:mTcID, TR':TrID, TC':TcID, S:State)
        = s (num (MTC, S))
    if MTC = next (TC') .
ceq e-num-move (MTC:mTcID, TR':TrID, TC':TcID, S:State) =
        p (num (MTC, S))
    if MTC = TC' .
ceq e-num-move (MTC:mTcID, TR':TrID, TC':TcID, S:State)
        = num (MTC, S)
    if not (MTC = TC' or MTC = next (TC')) .
```

1 つ目の等式は，move によって動いた列車 TR' の移動先の区間で，その区間の列車数が増加すること，2 つ目の観測は，列車が動く前にいた区間の列車数が減少することを意味している．3 つ目の等式は，それ以外の区間の列車数は，変化しないことを意味している．

最後に，move による pos および num の変化に関する等式を記述した．

3 章で提案した記法を適用することで，振舞遷移機械の定義を保ったまま，簡潔な仕様を記述することができた．

5.4　安全性の検証

一方通行の鉄道システムの安全性は，次のように表すことができる．

表明 5.1　任意の状態可能な状態 $u \in \Upsilon$ において，任意の列車 n と m (ただし，$n \neq m$)

が，同時に任意の区間 $t\ inT$ に進入しない．

$$\neg(pos_n(u) = t \wedge pos_m(u) = t)$$

5.4.1 原始隠蔽定数の宣言

モジュール CASES において，検証に使用する原始隠蔽定数を宣言した．

仕様と表明に現れる述語は，次の 2 つである．

- pos(C:TrID, S:State) = T:TcID
- green?(num(next(T:TcID), S:State))

これらのうち，変数 C と T を定数で具象化する．任意の列車を表す 2 つの定数 c1 と c2 と，任意の区間を表す t，t1，t2 と，t1 および t2 の次の区間を表す各定数 t1' と t2' を，次のように宣言した．

```
ops c1 c2 :  -> TrID
ops t t1 t2 t1' t2' -> TcID
eq next(t1) = t1' .
eq next(t2) = t2' .
```

以上の定数で具象化した各述語について，原子隠蔽定数を用意した．

5.4.2 検証を表現する項の定義

モジュール PROOF において，安全性検証を表現する項を定義した．

操作演算は，move 一つである．それに関する各場合について検証する項 MOVE を、以下のように定義した．

```
op MOVE : TrID TcID Case Case -> Bool
ceq MOVE(TR:TrID, TC:TcID, C:Case, C':Case) = true
    if comps(C, C') ⇒ (p (C o C') ⇒ p(move(TR, TC, (C o C')))) .
```

帰納段階において，定数で具象化した以下の操作演算について，検証を行う．

```
eq FORALL-ACTION(HYP:Case)
    = MOVE (c1, t1, c1t1 o gt1', HYP)
    | MOVE (c2, t2, c2t2 o gt2', HYP) .
```

```
mod CLAIM5-1 {
  ex(PROOF)
  eq p(S:State) =                                   (1) 論理和標準形に変換した表明 5.1
        (c1 = c2)
     ∨ (¬ (pos(c1, S) = t))
     ∨ (¬ (pos(c2, S) = t)) .

  eq  INI = p (init) .                              (2) 基底段階に関する検証指示

  ops hyp2 hyp3 : -> Case                           (3) 合成隠蔽定数による帰納法の仮定
  eq  hyp2 = ~c1t .
  eq  hyp3 = ~c2t .

  eq  IND = FORALL-ACTION (hyp2)                    (4) 帰納段階に関する検証指示
        | FORALL-ACTION (hyp3)  .
}

open CLAIM5-1                                       (5) c1 = c2 の場合の検証スクリプト
eq (c1 = c2) = true .
red INI | IND .
close

open CLAIM5-1                                       (6) c1 ≠ c2 の場合の検証スクリプト
eq (c1 = c2) = false .
red INI | IND .
close

-- reduce in %CLAIM5-1 : INI | IND                  (7) (6)の簡約結果
  MOVE(c1, t1, c1t1 o gt1', ~c1t)
| MOVE(c2, t2, c2t2 o gt2', ~c2t)
| true : Bool
(0.000 sec for parse,
 483 rewrites(0.040 sec), 1966 matches)
```

図 5.2: モジュール CLAIM5-1

5.4.3　表明 5.1 の検証

モジュール CLAIM5-1 は，モジュール PROOF で定義した帰納法を用いて，表明 5.1 を検証するためのモジュールである．

このモジュールを図 5.2 に示す．図 5.2 (1) で示した記述は，表明 5.1 を論理和標準形に変換したものである．(2) は基底段階に関する議論である．

次に，帰納法の仮定を隠蔽定数で表現する．論理和標準形に変換した表明 5.1 は，3 つの節から構成されている．しかし，最初の節は状態に関する言明でないため，合成隠蔽定数で表現することができない．最初の節には，別途場合分けを与えることとし (後述)，残りの 2 つの節について，それらを合成隠蔽定数で表現したものを，図 5.2 (3) に示すように記述した．最後に，帰納段階についての検証を，hyp2 と hyp3 の各場合について行うことを，図 5.2 (4) に示すように記述した．しかし，表明の最初の節についての場合分けが行われていない．そこで，図 5.2 (5) と (6) に示す証明譜を記述し，場合分けを行った．(5) は c1 = c2 の場合について，(6) は c1 = c2 でない場合について検証する．(5) の簡約結果は true であり，具体的な結果の記述については割愛する．(6) の簡約結果を，

```
mod CLAIM5-1 {
  ...
  eq  IND = FORALL-ACTION (hyp2 o  c2t)          (1) 帰納段階に関する検証指示
          | FORALL-ACTION (hyp2 o ~c2t)
          | FORALL-ACTION (hyp3 o  c1t)
          | FORALL-ACTION (hyp3 o ~c1t)  .
}

open CLAIM5-1                                    (2) c1 ≠ c2 の場合の検証スクリプト
eq (c1 = c2) = false .
red INI | IND .
close

-- reduce in %CLAIM1 : INI | IND}                (3) (2)の簡約結果
  MOVE(c1,t1,c1t1 o gt1',~c1t o c2t) ..... (a)
| MOVE(c2,t2,c2t2 o gt2',~c2t o c1t) ..... (b)
| true : Bool
(0.000 sec for parse,
 854 rewrites(0.060 sec), 2972 matches)}

red p(move(c1,t1,c1t1 o gt1' o ~c1t o c2t)) .    (4) (a) の場合について，簡約指示

-- reduce in %CLAIM1 :                           (5) (4)の簡約結果
      p(move(c1,t1,c1t1 o gt1' o ~c1t o c2t))
next(pos(c1,c1t1 o gt1' o ~c1t o c2t))
                          = t xor true : Bool
(0.000 sec for parse,
 52 rewrites(0.000 sec), 212 matches)
```

図 5.3: 場合分けを追加したモジュール CLAIM5-1

図 5.2 (7) に示す．(7) では，2 つの場合について，簡約が停止している．

そこで，hyp2 と hyp3 の各場合について，より詳細な場合分けを試みる．hyp2 について，hyp3 で使用している ~c2t とその否定 c2t について場合分けを行う．hyp3 についても，同様の場合分けを行う．この場合分けを追加したモジュール CLAIM5-1 を，図 5.3 に示す．図 5.3 (1) で示した帰納段階について，場合分けを追加している．

c1 = c2 でない場合について検証する証明譜 (図 5.3 (2)) を用いて，再度簡約を行った結果を図 5.3 (3) に示す．依然として検証は完了しないが，この結果から hyp2 o ~c2t と hyp2 o ~c1t の各場合については，検証が成功したことが読みとれる．ここで，検証が成功しない個々の場合 (a) と (b) について，簡約を試みる．(a) に関して簡約を行うコマンドを図 5.3 (4) に，その結果を図 5.3 (5) に示す．この結果は，列車 c1 が move(c1,t1,S) によって移動した区間 (つまり t1') と t の関係が不明であるため，簡約が停止したことを意味している．つまり，t1' = t について場合分けが必要と判明した．(b) でも同様の議論により，t2' = t について場合分けが必要である．

CLAIM5-1 において，これらの場合分けを行うため，使い捨ての原始隠蔽定数 t1'eqt, ~t1'eqt, t2'eqt および ~t2'eqt を定義する．これらを使い hyp2 o c2t と hyp3 o c1t の場合について，さらに詳細な場合分けを行う．

このようにして書き換えたモジュール CLAIM5-1 を図 5.4 に示す．図 5.4 (1) において，4 つの原始隠蔽定数を追加し，図 5.4 (2) において，追加した原始隠蔽定数を用いて場合

```
mod CLAIM5-1 {
  ops t1'eqt ~t1'eqt : -> Atom                                  (1) 追加した原始隠蔽定数
  ops t2'eqt ~t2'eqt : -> Atom

  ceq (next(pos(c1, ( t1'eqt o C:Case))) = t) = true
      if  comp( t1'eqt, C) .
  ceq (next(pos(c1, (~t1'eqt o C:Case))) = t) = false
      if  comp(~t1'eqt, C) .
  ceq (next(pos(c2, ( t2'eqt o C:Case))) = t) = true
      if  comp( t2'eqt, C) .
  ceq (next(pos(c2, (~t2'eqt o C:Case))) = t) = false
      if  comp(~t2'eqt, C) .

  eq  tbl ( t1'eqt) = ~t1'eqt .
  eq  tbl (~t1'eqt) =  t1'eqt .
  eq  tbl ( t2'eqt) = ~t2'eqt .
  eq  tbl (~t2'eqt) =  t2'eqt .

  eq  IND = FORALL-ACTION (hyp2 o  c2t o  t1'eqt)            (2) 帰納段階に関する検証指示
          | FORALL-ACTION (hyp2 o  c2t o ~t1'eqt)
          | FORALL-ACTION (hyp2 o ~c2t)
          | FORALL-ACTION (hyp3 o  c1t o  t2'eqt)
          | FORALL-ACTION (hyp3 o  c1t o ~t2'eqt)
          | FORALL-ACTION (hyp3 o ~c1t)  .
}

open CLAIM5-1                                                (3) c1 ≠ c2 の場合の
eq (c1 = c2) = false .                                           検証スクリプト
red INI | IND .
close

-- reduce in %CLAIM1 : INI | IND                             (4) (3)の簡約結果
  MOVE(c1,t1,c1t1 o gt1',~c1t o c2t o t1'eqt)
| MOVE(c2,t2,c2t2 o gt2',~c2t o c1t o t2'eqt)
| true : Bool
(0.000 sec for parse,
 1220 rewrites(0.140 sec), 4727 matches)
```

図 5.4: さらに場合分けを追加したモジュール CLAIM5-1

分けを行うように変更されている．

図 5.4 (3) の証明譜の簡約結果を図 5.4 (4) に示す．~c1t o c2t o t1'eqt の場合の検証が成功しない．この場合について簡約を行うと，false が得られ，表明が崩れていることがわかる．これを解決するには，補題が必要である．以下に補題 5.1 を示す．

補題 5.1　任意の状態可能な状態 $u \in \Upsilon$ において，任意の列車 n が，同時に任意の区間 $t \in T$ に存在するとき，その区間の信号 t は停止信号である．

$$pos_n(u) = t \Rightarrow \neg(green?(num_t(u)))$$

ここでは，この補題が証明できたものとして，議論を続ける (補題 5.2 を記述したコードは，付録 B.2.3 に示す)．~c1t o c2t o t1'eqt は，列車 c2 が t にいる場合に関する議論である．さらに，t1' と t は等しい場合についての議論である．従って，区間 t1' の信号は，補題 5.1 より停止信号である．同じことが，~c2t o c1t o t2'eqt についても言える．このため，これらの場合について，t1' および t2' の信号が停止信号である場合を示す原始隠蔽定数 ~gt1' と ~gt1' を加える．

帰納段階の議論に補題 5.1 を加えたモジュール CLAIM5-1 を図に示す．また，図 5.5

```
mod CLAIM5-1 {
  eq  IND = FORALL-ACTION (hyp2 o  c2t o  t1'eqt o ~gt1')
          | FORALL-ACTION (hyp2 o  c2t o ~t1'eqt)
          | FORALL-ACTION (hyp2 o ~c2t)
          | FORALL-ACTION (hyp3 o  c1t o  t2'eqt o ~gt2')
          | FORALL-ACTION (hyp3 o  c1t o ~t2'eqt)
          | FORALL-ACTION (hyp3 o ~c1t)  .
}

open CLAIM5-1
eq (c1 = c2) = false .
red INI | IND .
close

-- reduce in %CLAIM1 : INI | IND
true : Bool
(0.000 sec for parse,
 865 rewrites(0.140 sec), 3300 matches)
```

(1) 帰納段階に関する検証指示

(2) c1 ≠ c2 の場合の検証スクリプト

(3) (2)の簡約結果

図 5.5: 最終的なモジュール CLAIM5-1

(2) の証明譜を用いて簡約した結果を図 5.5 (3) に示す．以上の議論により，すべての場合について true が得られ，表明 5.1 が成立することが示された．□

第 6 章

単線区間の鉄道信号システム

本章では，より近代的な鉄道信号システムを取り上げ，3 章と 4 章で示した手法を適用する．この例題は，筆者が [33] で示したものに，改善を施したものである．

6.1 概要

自動閉そく式 (特殊)[2][3] は，列車の運行頻度が比較的低い単線区間に適用される閉そく方式である．単線用の信号システムは，前述したスタフ閉そくなどに見られるように様々な方式が使用されている．スタフ閉そくなどが，駅間の区間における列車同士の衝突防止を目的としているのに対し，自動閉そく式 (特殊) は，駅構内も含め，列車同士の衝突防止を目的とする．

本章では，自動閉そく式 (特殊) 方式を採用する鉄道システムにおいて，任意の隣接する 2 つの停車場とそれらを結ぶ単線の線路を取り出し，各停車場の線路配置などを抽象化した二隣接駅の鉄道システムを対象に議論を進める．

図 6.1 に，二隣接駅の鉄道システムを示す．tn $(n = 1, \ldots, 7, \mathsf{l}, \mathsf{r})$ は，区間の識別子で

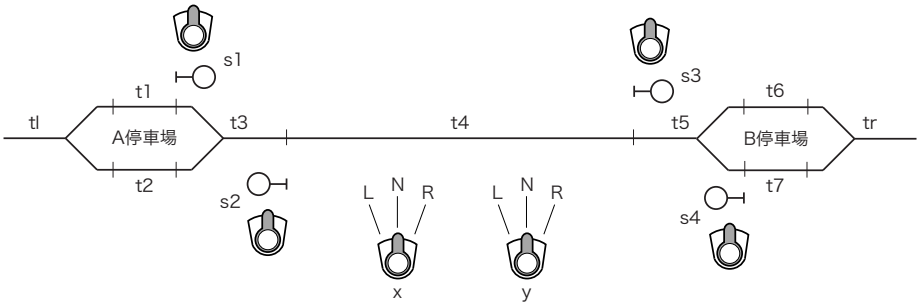


図 6.1: 二隣接駅の鉄道システム

ある．各区間には，その区間内に列車が存在するかどうかを調べるセンサーがある．A 停車場は，tl，t1，t2，t3 から，B 停車場は，t5，t6，t7，tr から構成される．tl や tr は，それらの停車場以遠の線路を含んでいる．t4 は，これらの停車場を結ぶ単線の線路である．t1 と t6 は，図の右方向へ走る列車が，t2 と t7 は，図の左方向へ走る列車が発着する区間である．

s1，s2，s3，s4 は信号の識別子である．各信号は，進行信号や停止信号を現示する信号機と，その信号機を制御するスイッチから構成される．スイッチを操作すると，後述する条件を満たす時，対応する信号機に進行信号が現示される (同時に分岐器も開通する)．s1 は t1 から t3 へ，s2 は t4 から t3 へ，s3 は t4 から t5 へ，s4 は t7 から t5 へ，進んでよいことを示す信号機である．今，t1 にいる右方向へ走行する列車が，s1 が現示する進行信号に従って t3 へ進入しようとしている．この列車が t3 に進入すると t3 のセンサーによって検知され，信号 s1 は停止信号に変化する．信号 s1 は，再度 s1 を操作するスイッチを操作しない限り，停止信号を保持する．他の信号機についても同様である．

スイッチ x と y は，t4 の運転方向を決めるための装置である．x は A 停車場に，y は B 停車場に設置されている．これらのスイッチは，l，n および r の 3 方向に操作できる．x と y が共に r のときは，t4 の運転方向は図の右方向に，共に l のときは，図の左方向に決定されており，それ以外の組み合わせは，運転方向が決定されていないものとする．

各区間のセンサー，各信号機および各スイッチの間には，以下に列挙する関係がある．

- x (または y) が n のとき，それを l (または r) に設定できるのは，y (または x) が n のときに限られる．
- x (または y) が n のとき，それを r (または l) に設定できるのは，y (または x) が r (または y) のときに限られる．
- x (または y) が r (または l) のとき，n に設定できるのは，スイッチ s1 (または s4) が使用中でないときに限られる．
- x (または y) が l (または r) のとき，n に設定できるのは，y (または x) が n のときに限られる．
- s1 (または s4) が使用中でないとき，使用中に設定できるのは，x と y が共に r (または l) かつ t3 と t4 (または t4 と t5) に共に列車が不在であり，かつ s2 (または s3) が使用中でないときに限られる．このとき，信号機 s1 (または s4) には進行信号が現示される．
- s2 (または s3) が使用中でないとき，使用中に設定できるのは，t3 と t2 (または t5 と t6) に列車が不在であり，かつ s1 (または s4) が使用中でないときに限られ

る．このとき，信号機 s2 (または s3) には進行信号が現示される．

- s1 や s2 (または s3 や s4) が使用中のとき，使用中でないに設定できるのは，t3 (または t5) に列車が不在であるときに限られる．このとき，信号機 s1 や s2 (または s3 や s4) には停止信号が現示される．
- t3 (または t5) に列車が進入したとき，信号機 s1 および s2 (または s3 および s4) には，停止信号が現示される．

二隣接駅の鉄道システムは，各区間 tn ($n = 1, \ldots, 7$) に，同時に 2 列車が進入しないことを保証する．

6.2　振舞遷移機械によるモデル化

二隣接駅の鉄道システムで使用するデータ型は，次の通りである．

- $C = \{0, 1, 2, \ldots\}$．列車の識別子．
- $T = \{\mathsf{tl}, \mathsf{t1}, \mathsf{t2}, \mathsf{t3}, \mathsf{t4}, \mathsf{t5}, \mathsf{t6}, \mathsf{t7}, \mathsf{tr}\}$．区間の識別子．
- $S = \{\mathsf{s1}, \mathsf{s2}, \mathsf{s3}, \mathsf{s4}\}$．信号機 (及びそれに付随するスイッチ) の識別子．
- $X = \{\mathsf{x}, \mathsf{y}\}$．区間 t4 の運転方向を決めるスイッチの識別子．
- $D_C = \{\mathsf{l}, \mathsf{r}\}$．列車の運転方向を示す値．
- $D_X = \{\mathsf{l}, \mathsf{n}, \mathsf{r}\}$．区間 t4 の運転方向を決めるスイッチの設定値．
- $N = \{0, 1, 2, \ldots\}$．自然数の集合．

これらのデータを用いて，二隣接駅の鉄道システムを表す振舞遷移機械 $\mathcal{S} = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ を定義する．二隣接駅の鉄道システムを表す状態空間 Υ の存在を仮定する．

観測 $\mathcal{O}$ は次のように表現できる．

- $\{pos_c : \Upsilon \to T \mid c \in C\}$．　列車 c の現在位置を区間の識別子 $t \in T$ で観測する．
- $\{dir_c : \Upsilon \to D_C \mid c \in C\}$．　列車 c の進行方向を方向の値 $d \in D_C$ で観測する．
- $\{num_t : \Upsilon \to N \mid t \in T\}$．　区間 t に存在する列車数を自然数 $n \in N$ で観測する．
- $\{which_x : \Upsilon \to D_X \mid x \in X\}$．区間 t4 における列車の運転方向を決めるスイッチ x に設定された方向を値 $d \in D_X$ で観測する．
- $\{using_s : \Upsilon \to \{\mathrm{true}, \mathrm{false}\} \mid s \in S\}$．信号機 s を制御するスイッチの様子を真偽値で観測する．true はスイッチ s が使用中，false は非使用中であることを表す．
- $\{sig_s : \Upsilon \to \{\mathrm{true}, \mathrm{false}\} \mid s \in S\}$．信号機 s の様子を真偽値で観測する．true は進行信号を，false は停止信号が現示されていることを表す．

初期状態 $\mathcal{I}$ は，任意の状態 $u \in \Upsilon$ に対し，次のように表現できる．

- $\forall c \in C.(dir_c(u) = \mathsf{r} \bigtriangledown dir_c(u) = \mathsf{l})$．列車は右方向または左方向へ走行する．
- $\forall c \in C.(dir_c(u) = \mathsf{r} \ \Rightarrow \ pos_c(u) = \mathsf{tl})$． 右方向へ走行する列車は，tl にいる．
- $\forall c \in C.(dir_c(u) = \mathsf{l} \ \Rightarrow \ pos_c(u) = \mathsf{tr})$． 左方向へ走行する列車は，tr にいる．
- $\forall t \in T.(num_t(u) = 0)$．各区間 t には，列車が存在しない．
- $\forall x \in X.(which_x(u) = \mathsf{n})$．区間 t4 の列車の運転方向を制御するスイッチ x は，共に中立 (n) である．
- $\forall s \in S.(using_s(u) = \text{false})$．信号機 s を制御するスイッチは，使用中ではない．
- $\forall s \in S.(sig_s(u) = \text{false})$．信号機 s は，停止信号を現示している．

ただし，$\bigtriangledown$ は排他的論理和とする．

遷移規則 $\mathcal{T}$ は，4 つの添字付き遷移規則の集合で表現する．ここでは，遷移規則が適用された時の状態を $u \in \Upsilon$，遷移規則が適用された後の状態を $u' \in \Upsilon$ と表記する．

- $\{move_{(c,t,d)} : \Upsilon \to \Upsilon \mid c \in C, t \in T, d \in D_C\}$．列車 c が区間 t から方向 d に隣接する区間へ移動することを表す遷移規則の集合である．この遷移規則の効果は次の通りである．列車の位置を表す pos_c のほか，各区間 t' (ただし，t' は区間 t と，t の方向 d 側に隣接する区間) の列車数を表す $num_{t'}$ が変化する．また，信号機を表す $sig_{s'}$ を変化させる遷移規則もある．効力条件については，自然語で解説した通りであり，解説は省略する．
- $\{turn_{(x,d)} : \Upsilon \to \Upsilon \mid x \in X, d \in D_X\}$．区間 t4 の運転方向を制御するスイッチ x に，方向 d を設定する．この遷移規則の効果は，$which_x$ の値が d に変化することである．効力条件については，自然語で解説した通りであり，解説は省略する．
- $\{allow_s : \Upsilon \to \Upsilon \mid s \in S\}$．信号機 s を制御するスイッチ x を使用中に設定する．この遷移規則の効果は，$using_s$ と $signal_s$ の値が true に変化することである．効力条件については，自然語で解説した通りであり，解説は省略する．
- $\{forbid_s : \Upsilon \to \Upsilon \mid s \in S\}$．信号機 s を制御するスイッチ x を非使用に設定する．この遷移規則の効果は，$using_s$ と $signal_s$ の値が false に変化することである．効力条件については，自然語で解説した通りであり，解説は省略する．

以上のように，二隣接駅の鉄道システムをモデル化した．

6.3　CafeOBJ による仕様記述

振舞遷移機械でモデル化した二隣接駅の鉄道システムを CafeOBJ で記述した．仕様は，全体で約 500 行であり，中規模な例題に属すると考えられる．完全なコードは，付録 C.1 に示す．

6.3.1　抽象データ型の記述

モジュール TRAINID

列車の識別子 (集合 C) を定義するモジュールである．CafeOBJ 組込みのモジュール NAT を輸入し，ソート名を TrID に名前替えした．また，TrID 上の項の等価性を判定する述語 = を定義した．

モジュール TRACKID

区間の識別子 (集合 T) を定義するモジュールである．可視ソート TcID 上に，区間の識別子を定義した．また，TcID 上の項の等価性を判定する述語 = を定義した．

モジュール SIGNALID

信号の識別子 (集合 S) を定義するモジュールである．可視ソート SgID 上に，信号の識別子を定義した．また，SgID 上の項の等価性を判定する述語 = を定義した．

モジュール LEVERID

t4 の運転方向を決めるスイッチの識別子 (集合 X) を定義するモジュールである．可視ソート LvID 上に，信号の識別子を定義した．また，LvID 上の項の等価性を判定する述語 = を定義した．

モジュール TRAINDIR

列車の運転方向 (集合 D_C) を定義するモジュールである．可視ソート TrDir 上に，値 l と r を宣言した．また，TrDir 上の項の等価性を判定する述語 = を定義した．

モジュール LEVERDIR

t4 の運転方向を決めるスイッチの意味する方向 (集合 D_X) を定義するモジュールである．可視ソート LvDir 上に，値 l，n および r を宣言した．また，LvDir 上の項の等価性を判定する述語 = を定義した．

6.3.2 抽象機械の記述

二隣接駅の鉄道システムを表す抽象機械は，モジュール ABB に記述した．仕様は付録 C.1 に示す．3 章で提案した記法を適用することで，振舞遷移機械の定義を保ったまま，簡潔な仕様を記述することができた．

6.4 安全性の検証

二隣接駅の鉄道システムの安全性は，7 つの表明で表す．また，これらを証明するための 36 の補題が必要である．二隣接駅の鉄道システムは，表明及び補題の数が多いので，最初に証明の概要を述べる．また，個々の証明に用いるモジュールは，付録に示した．

6.4.1 検証の概要

二隣接駅の鉄道システムの安全性は，以下に示す 表明 6.1 から 6.7 で表される．表明 6.n ($n = 1, \ldots 7$) は，各区間 tn において，列車の衝突がないことを述べている:

表明．6.1 から 6.7　二隣接駅の鉄道システムの安全性
任意の列車 c_1 と c_2 があり，到達可能な任意の状態 s について，

$$\neg(pos_{c_1}(s) = \mathsf{t}n \wedge pos_{c_2}(s) = \mathsf{t}n)$$

また，これらの表明を証明するために，多くの補題が必要である．補題は，3 つのグループに分けた．最初のグループ (A 群とする) は，表明 6.1，6.3，6.5 および 6.7 の証明に用いる補題である．次のグループ (B 群とする) は，A 群の補題と表明 6.3 および 6.5 を用いて証明でき，表明 6.2 および 6.6 の証明に用いる補題である．最後のグループ (C 群とする) は，表明 6.4 の証明に用いる補題である．

A 群の補題

A 群の補題は，表明 6.1，6.3，6.5 および 6.7 の証明に用いる．

補題 6.A.1 から補題 6.A.7 は，各区間 tn ($n = 1, \ldots 7$) において，ある列車が区間 tn に存在するとき，その区間の列車数を示す観測 $num_{\mathsf{t}n}$ は，1 以上であることを述べている:

補題 6.A.1 から補題 6.A.7　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t}n) \Rightarrow num_{\mathsf{t}n}(s) > 0$$

補題 6.A.8 から補題 6.A.11 は，各信号機 sn ($n = 1, \ldots 4$) において，信号機 sn が進行信号を現示しているならば，そのスイッチは使用中であることを述べている:

補題 6.A.8 から 6.A.11　到達可能な任意の状態 s について，

$$signal_{\mathsf{s}n}(s) \Rightarrow using_{\mathsf{s}n}(s)$$

補題 6.A.12 (または 補題 6.A.13) は，各信号機 s1 と s2 (または s3 と s4) について，どちらか一方しか進行信号を現示できないことを述べている:

補題 6.A.12　到達可能な任意の状態 s について，

$$(\neg signal_{\mathsf{s1}}(s)) \wedge signal_{\mathsf{s2}}(s)$$

補題 6.A.13　到達可能な任意の状態 s について，

$$(\neg signal_{\mathsf{s3}}(s)) \wedge signal_{\mathsf{s4}}(s)$$

補題 6.A.14 (または 補題 6.A.15) は，任意の列車が t3 (または t5) に存在するとき，信号機 s1 および s2 (または s3 および s4) が停止信号であることを述べている:

補題 6.A.14　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t3}) \Rightarrow (\neg signal_{\mathsf{s1}}(s) \wedge \neg signal_{\mathsf{s2}}(s))$$

補題 6.A.15　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t5}) \Rightarrow (\neg signal_{\mathsf{s3}}(s) \wedge \neg signal_{\mathsf{s4}}(s))$$

A 群の補題のうち，補題 6.A.1 から 6.A.11 までは，基本の場合分けを用いるだけで解くことができた．補題 6.A.12 と 6.A.13 では，基本の場合分けを用いたとき，2 つの場合が，補題 6.A.14 と 6.A.15 では 4 つの場合が残った．これらについては，補題を導入することで解決できた．

B 群の補題

B 群の補題は，表明 6.2 および表明 6.6 の証明に用いる．また，表明 6.3 および 6.5 を補題として用いる．

補題 6.B.1 (または 補題 6.B.2) は，任意の列車が t2 (または t6) に存在するとき，信号機 s2 (または s3) が停止信号であることを述べている:

補題 6.B.1　到達可能な任意の状態 s について，

$$(pos_{c}(s) = \mathsf{t2}) \Rightarrow (\neg signal_{\mathsf{s2}}(s))$$

補題 6.B.2　到達可能な任意の状態 s について，

$$(pos_{c}(s) = \mathsf{t6}) \Rightarrow (\neg signal_{\mathsf{s3}}(s))$$

補題 6.B.3 (または 補題 6.B.4) は，任意の列車 c1 が t2 (または t6) に存在し，かつ，左方向 (または右方向) に走る別の列車 c2 が t3 (または t5) に存在することはないと，述べている:

補題 6.B.3　到達可能な任意の状態 s について，

$$pos_{\mathsf{c1}}(s) = \mathsf{t2} \wedge pos_{\mathsf{c2}}(s) = \mathsf{t2} \wedge dir_{\mathsf{c2}}(s) = \mathsf{l}$$

補題 6.B.4　到達可能な任意の状態 s について，

$$pos_{\mathsf{c1}}(s) = \mathsf{t6} \wedge pos_{\mathsf{c2}}(s) = \mathsf{t5} \wedge dir_{\mathsf{c2}}(s) = \mathsf{r}$$

B 群の補題では，いずれも基本の場合分けを用いたとき，それぞれ 2 つの場合が残った．これらについては，補題を導入することで解決できた．

C 群の補題

C 群の補題は，表明 6.4 の証明に用いる．

補題 6.C.1(または 6.C.2) は，信号 s1(または s4) に進行信号が現示されているとき，t4 の運転方向は右 (または左) に決定されていることを述べている．

補題 6.C.1　到達可能な任意の状態 s について，

$$signal_{\mathsf{s1}}(s) \Rightarrow (which_{\mathsf{x}}(s) = \mathsf{r} \wedge which_{\mathsf{y}}(s) = \mathsf{r})$$

補題 6.C.2　到達可能な任意の状態 s について，

$$signal_{\mathsf{s4}}(s) \Rightarrow (which_{\mathsf{x}}(s) = \mathsf{l} \wedge which_{\mathsf{y}}(s) = \mathsf{l})$$

補題 6.C.3 は，信号 s1 および s4 は，同時には，どちらか一方しか進行信号が現示されないことを述べている．

補題 6.C.3　到達可能な任意の状態 s について，

$$(\neg signal_{\mathsf{s1}}(s)) \wedge signal_{\mathsf{s4}}(s)$$

補題 6.C.4(または補題 6.C.5) は，スイッチ x(または y) が r(または l) でないとき，信号 s1(または s4) には進行信号が現示されないことを述べている．

補題 6.C.4　到達可能な任意の状態 s について，

$$(\neg which_{\mathsf{x}}(s) = \mathsf{r}) \Rightarrow (\neg signal_{\mathsf{s1}}(s))$$

補題 6.C.5　到達可能な任意の状態 s について，

$$(\neg which_{\mathsf{y}}(s) = \mathsf{l}) \Rightarrow (\neg signal_{\mathsf{s4}}(s))$$

補題 6.C.6 (または補題 6.C.7) は，スイッチ s1(または s4) が使用中でないとき，信号 s1(または s4) には進行信号が現示されないことを述べている．

補題 6.C.6　到達可能な任意の状態 s について，

$$(\neg using_{\mathsf{s1}}(s)) \Rightarrow (\neg signal_{\mathsf{s1}}(s))$$

補題 6.C.7　到達可能な任意の状態 s について，

$$(\neg using_{\mathsf{s4}}(s)) \Rightarrow (\neg signal_{\mathsf{s4}}(s))$$

補題 6.C.8 (または補題 6.C.9) は，右方向 (または左方向) に走る列車が t3 (または t5) に存在するとき，スイッチ s1 (または s4) は使用中であることを述べている．

補題 6.C.8　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t3} \wedge dir_c(s) = \mathsf{r}) \Rightarrow using_{\mathsf{s1}}(s)$$

補題 6.C.9　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t5} \wedge dir_c(s) = \mathsf{l}) \Rightarrow using_{\mathsf{s4}}(s)$$

補題 6.C.10 (または補題 6.C.11) は，右方向 (または左方向) に走る列車が t3 (または t5) に存在するとき，スイッチ x (または y) は r または (l) であることを述べている．

補題 6.C.10　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t3} \wedge dir_c(s) = \mathsf{r}) \Rightarrow (which_{\mathsf{x}}(s) = \mathsf{r})$$

補題 6.C.11　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t5} \wedge dir_c(s) = \mathsf{l}) \Rightarrow (which_{\mathsf{y}}(s) = \mathsf{l})$$

補題 6.C.12 (または補題 6.C.13) は，右方向 (または左方向) に走る列車が t3 (または t5) に存在するとき，信号機 s4 (または s1) は停止信号であることを述べている．

補題 6.C.12　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t3} \wedge dir_c(s) = \mathsf{r}) \Rightarrow (\neg signal_{\mathsf{s4}}(s))$$

補題 6.C.13　到達可能な任意の状態 s について，

$$(pos_c(s) = \mathsf{t5} \wedge dir_c(s) = \mathsf{l}) \Rightarrow (\neg signal_{\mathsf{s1}}(s))$$

補題 6.C.14 は，列車が t4 に存在するとき，信号機 s1 および s4 は停止信号であることを述べている．

補題 6.C.14　到達可能な任意の状態 s について，

$$pos_c(s) = \mathsf{t4} \Rightarrow (\neg signal_{\mathsf{s1}}(s) \wedge \neg signal_{\mathsf{s4}}(s))$$

補題 6.C.15 は，右方向に走る列車 c1 が t3 に存在し，かつ左方向に走る別の列車 c2 が t5 に存在することはないと述べている．

補題 6.C.15　到達可能な任意の状態 s について，

$$\neg(pos_{\mathsf{c1}}(s) = \mathsf{t3} \wedge dir_{\mathsf{c1}}(s) = \mathsf{r} \wedge pos_{\mathsf{c2}}(s) = \mathsf{t5} \wedge dir_{\mathsf{c2}}(s) = \mathsf{l})$$

補題 6.C.16(または補題 6.C.17) は，列車 c1 が t4 に存在し，かつ右方向 (または左方向) に走る別の列車 c2 が t3(または t5) に存在することはないと述べている．

補題 6.C.16　到達可能な任意の状態 s について，

$$\neg(pos_{\mathsf{c1}}(s) = \mathsf{t4} \wedge pos_{\mathsf{c2}}(s) = \mathsf{t3} \wedge dir_{\mathsf{c2}}(s) = \mathsf{r})$$

補題 6.C.17　到達可能な任意の状態 s について，

$$\neg(pos_{\mathsf{c1}}(s) = \mathsf{t4} \wedge pos_{\mathsf{c2}}(s) = \mathsf{t5} \wedge dir_{\mathsf{c2}}(s) = \mathsf{l})$$

C 群の補題では，補題 6.C.6 と 6.C.7 は，基本の場合分けを用いることで解けた．補題 6.C.1，6.C.2，6.C.4 および 6.C.5 は，基本の場合分けを用いたとき，1 つの場合が簡約できずに残った．6.C.3，6.C.8，6.C.9，6.C.10，6.C.11，6.C.12，6.C.13 および 6.C.15 は 2 つ，6.C.16 では 3 つ，6.C.14 では 4 つが残った．これらについては，いずれも補題を導入することで解決できた．

6.4.2　検証の結果

二隣接駅の鉄道システムが表明 6.1 から 6.7 に示す性質を有することを証明する．

表 6.1: 基本の場合分けを用いた時の残存場合数

表明または補題名	検査した場合数	残存場合数
表明 6.1	104	2
表明 6.2	104	2
表明 6.3	104	4
表明 6.4	104	4
補題 6.A.12	68	2
補題 6.A.14	68	4
補題 6.B.1	68	2
補題 6.B.3	156	2
補題 6.C.1	68	1
補題 6.C.3	68	2
補題 6.C.4	68	1
補題 6.C.8	102	2
補題 6.C.10	102	2
補題 6.C.12	102	2
補題 6.C.14	68	4
補題 6.C.15	208	2
補題 6.C.16	156	3

証明 付録 C.2 に示す各検証モジュールを CafeOBJ の処理系に読み込ませ，各表明および補題を表現する項を簡約することで，検証できる．□

以上の議論により，二隣接駅の鉄道システムが所望の安全性を有することが証明できた．

前述した各補題について，基本の場合分けを用いたときの，検査した場合数と，`true` まで簡約できずに残存した場合数を，表 6.1に示す．なお，表中では，残存数 0 の補題と，ある補題と対になっている補題については，記載を省略した．表 6.1は，基本の場合分けが，効果的に各場合を検査し，検証者が考慮しなければならない場合を大幅に減少させていることを示している．

また，付録で示すように，すべての各表明または補題は図 4.1で示したひな型を用いることで記述できた．また，必要に応じて，場合分けの追加や補題の導入を容易に行うことができた．このことは，本手法の有効性を実証していると考えられる．

6.4.3 場合分けの自動生成の結果

4.3 で示した場合分けの自動生成の手法を，基本の場合分けで検証が完了しない各場合について適用した．その結果をグループ A の補題について示すが，他のグループの補題についても，同様の結果が得られた．

補題 6.A.12 と補題 6.A.13

基本の場合分けで簡約できなかった場合を，次に示す．

```
ALLOW(s1,txr o tyr o ~t3nz o ~t4nz o ~us1 o ~us2, s1r)
ALLOW(s2,~t2nz o ~t3nz o ~us1 o ~us2,s2r)
```

最初に，前者の場合について述べる．相補対 s2g と~s2g を用いて場合を生成し，操作演算 allow(s1,s) が補題を保存することを確かめる．このとき，s2g (信号 s2 が進行信号) を加えた場合が偽となることがわかる．しかし，~us2 (信号 s2 のスイッチはオフ) が効力条件に表れている．これらの結果から，スイッチが真でないのに，進行信号が現示されている場合を解析していることがわかる．この場合が到達不能であることを，補題 6.A.9 によって示すことができる．次に，後者の場合について述べる．相補対 s1g と~s1g を用いて場合を生成し，操作演算 allow(s2,s) が補題を保存することを確かめる．前者の場合と同様の結果が得られ，到達不能であることを，補題 6.A.8 によって示すことができる．

同様に，補題 6.A.13 では，補題 6.A.10 と補題 6.A.11 が導かれる．

補題 6.A.14

基本の場合分けで簡約できなかった場合を，次に示す．

```
MOVE(c1,t1,r,c1t1 o c1r o s1g,~c1t3)
MOVE(c1,t4,l,c1t4 o c1l o s2g,~c1t3)
ALLOW(s1,txr o tyr o ~t3nz o ~t4nz o ~us1 o ~us2,s1r o s2r)
ALLOW(s2,~t2nz o ~t3nz o ~us1 o ~us2,s1r o s2r)
```

最初の 2 つの場合 (帰納法の仮定が ~c1t3) について述べる．相補対 s2g と~s2g (または s1g と~s1g) を用いて場合を生成し，操作演算 move(c1,t1,r,s)(または move(c1,t4,l,s)) が補題を保存することを確かめる．このとき，s2g (または s1g) を

加えた場合が偽となることがわかる．これらはいずれも，信号 s1 と s2 が同時に進行信号になった場合を検証しようとして失敗している．この結果から，補題 6.A.12 が導かれる．

同様に，補題 6.A.15 では，補題 6.A.13 が導かれる．

次に，残りの 2 つの場合 (帰納法の仮定が `s1r o s2r`) について述べる．いずれも列車 c1 の位置に関する原始隠蔽定数の相補対を用いて場合を生成し，操作演算 `allow(s1,s)`(または `allow(s2,s)`) が補題を保存することを確かめる．このとき，`c1t3` (または `c1t2`) を加えた場合が偽となることがわかる．これらはいずれも，それらの区間に列車がいないことを表している`~t3nz` (または `~t2nz`) と矛盾している．この結果から，補題 6.A.3 (または補題 6.A.2) が導かれる．

同様に，補題 6.A.15 では，補題 6.A.5 (または補題 6.A.6) が導かれる．

第 7 章

結論

本章では，まとめとして，本論文で提案した仕様記述法，検証手法に対する考察と，今後の課題を述べる．最後に，関連研究について述べ，最終的な結論を述べる．

7.1 考察および今後の課題

7.1.1 抽象機械の記述法の比較

これまで，振舞遷移機械に基づき，CafeOBJ の仕様を記述した事例は [29][30] などに見られる．これらでは，概ね，次のような記述法を取っていた．o，τ，$e_{(\tau,o)}$ および c_τ は，それぞれ，振舞遷移機械の観測，遷移規則，効果 (τ による o の変化) および τ の効力条件に対応する CafeOBJ 上の表現である．隠蔽ソート H は，状態空間 Υ を表現しており，h は H 上の変数である．これらに対して，次の 2 つの等式を与える．

`ceq` $o(\tau(h))$ `=` $e_{(\tau,o)}(h)$ `if` $c_\tau(h)$.

`ceq` $o(\tau(h))$ `=` $o(h)$ `if` $\neg(c_\tau(h))$.

最初の等式は，τ が適用されたとき，効力条件 c_τ が真のとき，効果 $e_{(\tau,o)}$ に従って o が変化することを意味している．次の等式は，効力条件 c_τ が偽のとき，o が変化しないことを意味している．

添字付きの観測や遷移規則は，無限状態を持つ状態機械の表現に欠かせない．上記の記述法では，添字付きの観測や遷移規則を扱うことは考慮されていないため，そのような観測や遷移規則を持つ振舞遷移機械を記述する際には，次のような問題が発生した．

例えば，遷移規則 $move_{(c,\mathsf{t1},\mathsf{r})}$ の観測 pos_c の変化を表現するには，次の 2 つの等式を記述することになる:

`ceq pos(`c_1`, move(`c_2`,` t`,` d`,` h`)) = t3`

```
        if c1 = c2 ∧ t = t1 ∧ d = r
            ∧ pos(c2, h) = t1 ∧ dir(c2, h) = r ∧ staff(h) = c .
    ceq pos(c1, move(c2, t, d, h)) = pos(c1, h)
        if ¬(c1 = c2 ∧ t = t1 ∧ d = r
            ∧ pos(c2, h) = t1 ∧ dir(c2, h) = r ∧ staff(h) = c) .
```

または，操作演算のアリティ部分に具体的な定数を当てはめ，次のように記述することになる．

```
    ceq pos(c1, move(c2, t1, r, h)) = t3
        if c1 = c2
            ∧ pos(c2, h) = t1 ∧ dir(c2, h) = r ∧ staff(h) = c .
    ceq pos(c1, move(c2, t1, r, h)) = pos(c1, h)
        if ¬(c1 = c2
            ∧ pos(c2, h) = t1 ∧ dir(c2, h) = r ∧ staff(h) = c) .
```

$move_{(c,\mathsf{t1},\mathsf{r})}$ の効力条件は，$pos_c = \mathsf{t1} \wedge dir_c = \mathsf{r} \wedge \mathit{staff} = c$ である．しかし，条件部分には，効力条件以外に，識別子が所望のものかどうかを判定するための条件式が現れ，煩雑になっている．

また，5 章と 6 章で取り上げた例題に見られるように，複数の観測に効果が及ぶときの記述は，より簡潔になる．以下は，5 章の例題を用いて，比較のために，従来手法を用いて，move による num の変化を記述したものである．

```
    ceq num(t1, move(c2, t2, h)) = p(num(t1, h))
        if t1 = t2
            ∧ pos(c2, h) = t2 ∧ green?(num(next(t1), h)).
    ceq num(t1, move(c2, t2, h)) = s(num(t1, h))
        if t1 = next(t2)
            ∧ pos(c2, h) = t2 ∧ green?(num(next(t1), h)).
    ceq pos(c1, move(c2, t1, r, h)) = pos(c1, h)
        if ¬(c1 = c2 ∨t1 = next(t2)
            ∧ pos(c2, h) = t1 ∧ dir(c2, h) = r ∧ staff(h) = c) .
```

これに対して，本論文で提案した記法は，添字を考慮していることと，振舞遷移機械の遷移規則の定義で用いた，効力条件と効果を表現するための演算を導入することで，等式宣言の条件部分において，添字に関する条件と効力条件を分離して記述することができる．記述はやや長くなるものの，振舞遷移機械と一対一に対応する仕様を記述できる．

7.1.2 安全性検証の手法

本論文では，検証において扱うべき場合を項で表現するために，合成隠蔽定数という新たな手法を提案した．合成隠蔽定数は，検証に必要な等式を制御するための技術であり，CafeOBJ が元々持っている機能だけで実現可能である．

合成隠蔽定数のアプリケーションとして，帰納法による安全性検証を選び，仕様の構文的な情報のみに基づき，マトリクス状に場合を組織し，マトリクスを再利用しながら，複数の性質の検証が効率良く行えることを，実例に基づいて示した．

このマトリクスには他にも次の利点があると考えられる．追加の場合分けを行ったとき，場合分けの記述が一箇所に集中しているため，どのような場合分けを行ったかが，一目瞭然となる．例えば，s_{hyp} に関する場合について，述語 p による場合分けを行うには，次のように記述した:

```
| FORALL-ACTION(s_hyp o s_p)
| FORALL-ACTION(s_hyp o s_¬p)
```

こうした特徴は，場合分けの漏れを防ぐために有効であると考えられる．

合成隠蔽定数のもう一つのアプリケーションとして，隠蔽定数を組み合わせた場合の生成を提案した．本論文で取り上げた例題では，この手法により証明し切れるものはなかったが，原理的には，この手法の適用により，自動的に証明可能な問題が増えることが期待される．また，場合分けの候補として得られた隠蔽定数を用い，表明が偽になる場合を探すことによって補題を推測できたことを実例に基づいて示した．

提案した手法では，自明な検証は計算機に任せ，本質的に難しい箇所にだけ集中できるようにすることによって，検証者を支援できた．また，合成隠蔽定数の手法は，今まで行われていた処理系と対話する検証手法でも用いることができる．このため，本手法で検証に行き詰まったときに，これまで用いられてきたあらゆる手法を動員し，問題の解決にあたれると期待できる．

本手法では，一度に多くの書換えを行うため，その効率が問題になる．本研究では，Frantz Allegro Common Lisp によってコンパイルした CafeOBJ インタプリタを用い，512MB のメモリを搭載した Apple PowerBook G4 (PowerPC G4 500MHz) において実行させた．本論文で取り上げた例題については，1 つの表明を証明するのに最大で 1 分程度の時間を要した．検証に不要な書換えを減らすことで検証時間を短縮するなど，本手法は改善の余地がある．

本研究において，合成隠蔽定数の実現方法と，それが実際の問題において適切に応用で

きることを，実例に基づいて示した．しかし，合成隠蔽定数に関する理論的研究はまだ行っていない．本論文で示したように，合成隠蔽定数は興味深い応用が可能であるので，その理論的研究は十分に価値があるものと期待でき，今後の課題とする．

また，本論文では，安全性のみを取り上げたが，振舞遷移機械の重要な性質として，他に活性や振舞等価性がある．これらの性質に検証に，合成隠蔽定数を応用し，同様に再利用可能なフレームワークを含む検証の組織化を行うことは，十分に期待できるため，今後の課題として興味深い．

7.2 関連研究

本節では，関連研究を，状態機械の安全性検証，代数仕様における検証支援および鉄道分野における形式手法の適用事例の 3 つの観点から取り上げ，本研究との比較を行う．

7.2.1 状態機械の安全性検証

有限状態機械の安全性検証は，モデル検査法 [10][22] が最も有望であると考えられる．モデル検査法は，検証したい性質を時相論理で記述することで，自動的な検証を可能としている．しかし，状態数に対して線形の計算時間を使うこと，他にも膨大な計算資源が必要であることが問題点として挙げられる．このため，たとえ有限状態機械であっても，巨大なシステムの検証にモデル検査法を使うのは難しいと考えられる．

本研究で用いた振舞遷移機械を含め，Pnueli の FTS (Fair Transition Systems)[23] や Lynch の I/O-オートマトン [21] は，無限状態機械を対象とした状態機械のモデルである．これらのうち，Garland と Lynch による IOA-Toolkit[37] という検証器がある．IOA-Toolkit は，Garland の Larch[12] を使い，I/O-オートマトンでモデル化した状態機械に関する性質を検証するための検証支援系である．ユーザは帰納法の戦略や場合分けの指示を与えることによって，検証を進めることができる．

IOA-Toolkit は検証に必要な補題を探すツール Daikon[16][39] を持つ．ユーザは具体的な実行系列を与える．Daikon はその実行を生成し，一般化やヒューリスティクスを用いていくつかの不変式を提案する．提案された不変式は，有限の実行系列から推測されたものであるので，必ず成立するとは限らず，別途検証が必要である．ユーザは提案された不変式から補題として使えそうなものを選び，それが成立することを検証したり，他の検証において補題として用いる．一方，本研究で提案した場合を生成する手法は，表明が偽になる場合を探すため，それが到達不能であるという補題と見なすこともできる．提案した手法では場合に基づいて補題を推測しており，表明が成り立つならばこの補題も成り立

つと予想できる点において優れている．一方で，場合分けに用いる原始隠蔽定数の相補対が事前に判明していなければ手法そのものが使えない点においては劣っていると考えられる．

7.2.2 代数仕様における検証支援

本研究は，CafeOBJ 自身を用いた検証支援であると見ることができる．CafeOBJ を含む代数仕様言語における検証支援については，[15] がある．Goguen らは，WWW (World Wide Web) アプリケーションとして，インターネット経由で複数の仕様記述者や検証者が共同で作業できるシステム TATAMI を実装・提案している．TATAMI のもう一つの特徴は，画像などを用いて，検証を可視化できることである．この中で，抽象データ型に関する帰納法の検証と，隠蔽代数に基づく状態機械について振舞等価性に関する検証を行っており，[13] で実際にシステムを利用できる．本研究で使用した仕様も同じ隠蔽代数に基づく状態機械を記述している．

本研究で用いた振舞遷移機械は振舞等価性に関する定義を含まないが，振舞遷移機械はこれを含める定義に既に拡張されている [30]．振舞等価性の議論は，観測操作の構造に関する帰納法，余帰納法 [14]，テスト集合余帰納法 [24] が提案されている．これらの検証手法に対しても，合成隠蔽定数の手法は適用可能であることが予想される．また，本研究で示した手法は，マトリクスなど視覚的にわかりやすい構造を持っているので，TATAMI などのツールを用いて視覚化することで，直感的にわかりやすい表現をとることは，容易であると考えられる．

森らは，無限状態を持つ振舞仕様の性質を検証する手法として，振舞モデル検査 [26] を提案している．振舞モデル検査は主に導出原理が用いられ，CafeOBJ 処理系を拡張し実装されている．このため，項書換え系の利点であった検証の読みやすさや，処理の軽さが失われている．

松本らは，コンポーネントに形式手法を与え，それらを組み合わせることであるシステム形式仕様を開発し，詳細化検証を行うツールを実装・提案している [25]．この中で，モデルを工夫することにより，検証において帰納法が必要になる部分を減らすことによって，自動化できる検証を増やしている．しかし，帰納法を完全に排除できないため，場合分けや補題の発見について，人の介在を要する．この中で，コンポーネントは [18] で提案された射影演算を用いて記述されている．射影演算は，隠蔽代数に基づく抽象機械の記述を再利用する手法であり，巨大なシステムを記述する際に期待されている．本研究を含め，振舞遷移機械では射影演算を用いた事例がないため，今後の課題である．

Bouhoula は検証器 SPIKE [6] を用いて，状態代数に基づく抽象機械の詳細化検証 [7] を行っている．SPIKE は書換え帰納法に基づく検証器であり，書換え帰納法は帰納法を自動化する手法として有望視されている．SPIKE では，帰納法と背理法を行うことができ，場合分けを半自動的に行い，検証を進めることができる．しかし，SPIKE で用いている手法は，場合分けを行うために処理系に手を入れ特殊な書換え戦略を導入した．このためこの手法を CafeOBJ に取り入れることは困難であると予想される．また，[7] では，小規模な例題 (スタックや集合) に対して適用しているが，現実の例題について適用できるかどうかは未知である．

7.2.3 鉄道分野における形式手法

鉄道は言うまでもなく，ひとたび事故が起これば人命が危険にさらされる高信頼システムである．鉄道における様々な技術的要素に対して，形式手法の適用が試みられている．

特に，鉄道側の要請が高いのは，連動装置の論理設計である．連動装置は，駅構内の分岐器と信号機の動作を制御する装置である．信号は，進路に対して 1 つ設けられ，その進路が開通していることを保証する．進路が開通しているとは，進路場にある分岐器が正しい方向に開通しており，その区間に列車がいないことを言う．信号機の現示に対して，分岐器を正しく動かす装置が連動装置である．連動装置の設計では，特定の駅について議論を行えば済むため，有限機械としてモデル化し，モデル検査法による検証が行われている．HOL に基づくモデル検査法の適用として，[27] がある．モデル検査器 SPIN[17] を用いた事例として [9] ，SMV[10] を用いた事例として [19] がある．しかし，いずれの手法でも，モデル検査は膨大な計算資源を必要とするため，上野駅や東京駅クラスの巨大なターミナルの検証には成功していない．

一方，本論文では，列車同士の衝突を防ぐためのスキームを記述したと考えることができる．このスキームをある特定の駅について具象化し，有限状態機械で表現したものが前述の研究であると見ることができる．日本鉄道電気技術協会は，1996 年に新たな閉そく方式として電子タブレット閉そく [28] を提案したが，安全性については，具象化した特定の駅について議論を行ったのみであった．新たな閉そく方式に対しては，その方式を適用したあらゆる事例について安全性を保証できることが望ましいと言える．

Bjørner らは，中国の鉄道を対象に，形式仕様を記述している．これらのうち，列車ダイヤの作成，ダイヤが乱れた場合の，ダイヤ再作成などのため，仕様とそれに基づくツールを実装している [5]．Bjørner らは，鉄道分野固有のデータ型を整理しており，こうした成果を取り入れることは有益であると考えられる．巨大なターミナルを表現できるスキー

ムを記述し，その安全性を検証することで，その具体例である上野駅や東京駅の安全性について保証できると期待される．

7.3 まとめ

本論文では，振舞遷移機械に基づく安全性検証に焦点を当て，仕様記述と検証過程について，次の提案を行った．仕様記述においては，振舞遷移機械と親和性の良い記述スタイルを提案した．検証過程においては，既存の CafeOBJ 処理系を用いて，場合分けに関する支援を行うための 3 つの手法を提案した．第一の手法は，場合を項の組み合わせで表すための手法である．第二の手法は，場合分けをマトリクス状に整理する手法である．第三の手法は，第二の手法で場合分けを尽くせなかったときに，場合分けに有効な候補を見つける手法である．これらの手法を用いて，自明な検証は計算機に任せ，本質的に難しい箇所にだけ集中できるようにすることによって，検証者を支援できた。本論文では，実際に使用されている鉄道信号システムを例題として取り上げ，これらの手法を安全性検証に適用し，支援効果があることを確かめた．

本論文は，以下に示す意義がある．

仕様記述においては，振舞遷移機械を CafeOBJ で記述することが以前に比較して容易になり，仕様の可読性も向上した．このことは，振舞遷移機械でモデル化したシステムの性質を検証する際に，計算機による支援が受けやすくなり，かつ，コミュニケーションの道具としての仕様の質が向上したことを意味する．

検証においては，安全性検証における場合分けを組織化し，場合分けの漏れや見落としを防げる可能性を向上させ，検証者を支援できた．基本の場合分けは表明毎に再利用できるため，これまでより効率的に検証できる．基本の場合分けで議論が尽くせなかった部分については，簡単に場合分けを追加することができる．それらは集中的に記述されるので，どんな場合分けを行ったのかが明瞭に分かり，場合分けの漏れを防ぐ効果があると考えられる．これらの手法は，CafeOBJ 処理系だけでなく，項書換えに基づく他の検証支援系を用いても，簡単に実現可能であると考えられる．

本論文で取り上げた事例は，実際に使用されている鉄道システムである．これは，振舞遷移機械や CafeOBJ が現実的な問題に適用でき，簡約のみによって推論可能であることを示した例としても貴重である．

謝辞

本研究を行なうに当たり，終始御指導を賜わった二木 厚吉教授に深く謝致します．

本論文の審査をしていただき，有益な御助言をいただきました，本学情報科学研究科落水浩一郎教授，片山卓也教授，日比野靖教授，東京大学大学院玉井哲雄教授に感謝致します．

また，日頃から有益な御助言をいただき，多面に渡って励ましていただいた本学客員研究員 緒方 和博 博士に感謝致します．

最後に，本論文をまとめるに当たって御協力いただいた言語設計学講座の諸兄に厚く御礼申し上げます．

参考文献

[1] CafeOBJ web page. `http://www.ldl.jaist.ac.jp/cafeobj/`.

[2] 普通鉄道構造規則. 国土交通省令第 14 号.

[3] 鉄道運転規則. 国土交通省令第 15 号.

[4] Baader, F. and Nipkow, R.: *Term rewriting and all that.* Cambridge University Press, 1999.

[5] Bjørner, D., George, C., and Prehn, S.: Secheduling and Rescheduling of Trains. In Hinchey, M., and Bowen, J. eds: "Industrial-strength formal method in practice", Springer-Verlag, pp. 157–184, 1999.

[6] Bouhoula, A.: Automated theorem proving by test set induction. *Journal of Symbolic Computation.* Vol. 23, pp. 47–77. 1997.

[7] Bouhoula, A. and Rusinowitch, M.: Observational proofs by rewriting. *Theoretical Computer Science.* Vol. 275, No. 1-2. pp. 675–698. 2002.

[8] Chandy, K. and Misra, J.: *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[9] Cimatti, A., Giunchiglia, F., Mongardi, G., Romano, D., Torielli, F., and Traverso, P.: Model Checking Safety Critical Software with SPIN: an Application to a Railway Interlocking System. In *Proceedings of the Seventeenth International Conference on Computer Safety, Reliability and Security (SAFECOMP'98).* Heidelberg, Germany. 1998.

[10] Clarke, E. Grumberg, O. and Peled D: *Model Checking.* MIT Press, 1999.

[11] Diaconescu, R. and Futatsugi, K.: *CafeOBJ report.* Number 6 in AMAST Series in Computing. World Scientific, 1998.

[12] Garland, S., Guttag, J. and Horning, J.: An overview of Larch. In *Functional programming, Concurrency, Simulation and Automated Reasoning*, LNCS 693. Speinger-Verlag. pp. 329–438. 1993.

[13] Goguen, J.: http://www.cs.ucsd.edu/groups/tatami/.

[14] Goguen, J: and Malcolm, G.: A hidden agenda. (245):55–101, 2000.

[15] Goguen, J. and Lin, K.: Web based support for cooperative Software Engineering. In *Proceedings of International Symposium on Multimedia Software Engineering.* ed. Tsai, J. and Chuang, P. IEEE Press. pp. 25–32 2000.

[16] Ernst, M., Cokrell, J. Grisworld, W. and Notkin, D.: Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, Vol. 27, No. 2. pp. 1-25. 2001.

[17] Holzmann, G.: The Model Checker SPIN. *IEEE Transactionon Software Engineering*, No. 5, Vol. 23 pp. 279–295, 1997.

[18] Iida, S., Diaconescu, R., Futatsugi, K., and Lucanu, D.: Concurrent object composition in CafeOBJ. Technical Report, IS-RR-98-0009S, JAIST, 1998.

[19] 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 (平成 10 年度後期) 全国大会, Vol. 1, pp. 240–241. 1998.

[20] Futatsugi, K.: Formal Methods in CafeOBJ. Lecture Notes in Computer Science, 2441, Springer, pp.1-20, 2002. (invited paper)

[21] Lynch, N.: Distributed Algorithms. Morgan Kaufmann Publishes, Inc., 1996.

[22] Magee, J. and Kramer, J.: *Concurrency.* John Wiley & Sons, 1998.

[23] Manna, Z. and Pnueli, A.: *The temporal logic of reactive and concurrent systems: specification.* Springer-Verlag, 1991.

[24] 松本 充広, 二木 厚吉 : テスト集合余帰納法を用いた振舞等式の検証. コンピュータソフトウェア, Vol.19, No.1, 2002.

[25] Matsumoto, M., and Futatsugi, K. : The support tool for highly reliable component-based software development. In *proceedings of APSEC'2000*, IEEE CS Press, pp. 35–43, 2000.

[26] Mori, A. and Futatsugi, K.: CafeOBJ as a tool for behavioral system verification. in *Symposium on Software Security 2002* (to be published as a Springer LNCS), 2002. (to apear)

[27] Morley, M.: Safety Assurance in Interlocking Design. Ph.d thesis. Univ. of Edinburgh. 1996.

[28] 日本鉄道電気技術協会: ローカル鉄道向け単線閉そく装置の開発事業研究報告書, http://prg1.nippon-foundation.or.jp/JIGYOU/1996/NFPRG96.nsf/V_Category/F2E9752FC23D90E14925655B00271D5B?OpenDocument.

[29] Ogata, K. and Futatsugi, K.: Modeling and verification of distributed real-time systems based on CafeOBJ. In ASE'01. IEEE CS Press. pp. 185–192. 2001.

[30] Ogata, K. and Futatsugi, K.: Formal analysis of Suzuki&Kasami distributed mutual exclusion algorithm. In FMOODS'02. Kluwer Academic Publishers. pp. 181–195. 2002.

[31] Ogata. K. and Futatsugi K.: Formal analysis of the iKP electric payment protocols. In Proc. of Symposium on Software Security 2002 (to be published as a Springer LNCS). 2002. (to apear)

[32] Spivey, J.: Understanding Z: A specification language and its formal semantics. Cambridge University Press. 1988.

[33] Seino, T., Ogata, K., and Futatsugi, K.: Specification and verification of a single-track railroad signaling in CafeOBJ. *IEICE TRANSACTIONS on Fundamental of Electronics, Communications and Computer Science*, Vol. E84-A, No.6, pp. 1471–1478, 2001.

[34] Seino, T., Ogata, K., and Futatsugi, K.: Specification and verification of tokenless blocking systems with CafeOBJ. In *Proceedings of ITC-CSCC 2001.* Vol. II, pp. 807–811, 2001.

[35] 清野 貴博, 緒方 和博, 二木 厚吉: 項書換えを用いた安全性検証の組織化. ソフトウェア工学の基礎 IX 日本ソフトウェア科学会 FOSE 2002. 近代科学社, Vol. 28, pp. 107–118. 2002.

[36] 清野 貴博, 二木 厚吉: 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 – 鉄道信号システムの形式仕様とその検証 –. 情報処理学会第 57 回 (平成 10 年度後期) 全国大会, Vol. 1, pp. 236–237. 1998.

[37] Soegaard-Andersen, J., Garland, S., Guttag, J. and Lynch, N.: Computer-assisted simulation proofs. In *CAV'93.* LNCS 697. Springer-Verlag. pp. 305–319. 1993.

[38] Yoshitake, I. and Akimoto, A.: *An explanation of facilities for driving security.* Japan Railway Books inc., 1984. (in Japanese).

[39] Win, T., Ernst, M., Garland, S., Kirli, D. and Lynch, N.: Using simulated execution in verifying distributed algorithms. In *Verification, Model checking, and Abstract Interpretation*, LNCS 2575. Speinger-Verlag. pp. 283–297. 2003.

[40] Woodman, M. and Heal, B: Introduction to VDM. McGraw-Hill. International series in software engineering. 1993.

本研究に関する発表論文

査読付きの発表論文

[1] 清野 貴博, 緒方 和博, 二木 厚吉: 項書換えを用いた安全性検証の組織化. コンピュータソフトウェア, 日本ソフトウェア科学会. (投稿中)

[2] 清野 貴博, 緒方 和博, 二木 厚吉: 項書換えを用いた安全性検証の組織化. ソフトウェア工学の基礎 IX 日本ソフトウェア科学会 FOSE 2002. 近代科学社, Vol. 28, pp. 107–118. 2002.

[3] 清野 貴博, 緒方 和博, 二木 厚吉, 日比野 靖: 実時間制約を考慮したマルチタスキングのモデル化. ソフトウェア工学の基礎 VIII 日本ソフトウェア科学会 FOSE 2001. 近代科学社, Vol. 26, pp. 143–146. 2001.

[4] Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi: Specification and verification of a single-track railroad signaling in CafeOBJ. *IEICE TRANSACTIONS on Fundamental of Electronics, Communications and Computer Science*, Vol. E84-A, No.6, pp. 1471–1478, 2001.

[5] Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi: Specification and verification of tokenless blocking systems with CafeOBJ. In *Proceedings of ITC-CSCC 2001*. Vol. II, pp. 807–811, 2001.

[6] Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi: Specification and verification of a single-track railroad signaling in CafeOBJ. In *Proceedings of ITC-CSCC 2000*, Vol. 1, pp. 268–273. 2000.

[7] 清野 貴博, 緒方 和博, 二木 厚吉: 代数仕様言語 CafeOBJ による鉄道信号システムの記述と検証. ソフトウェア工学の基礎 VI 日本ソフトウェア科学会 FOSE '99, 近代科学社, Vol. 22, pp. 180–187. 1999.

査読なしの発表論文

[8] 清野 貴博, 緒方 和博, 二木 厚吉: 代数仕様言語 CafeOBJ による実時間システムの仕様記述と検証の一例– timed two-process race の仕様記述と検証 –. 信学技報, Vol. 100, No. 569, pp. 9–16. 2001.

[9] 清野 貴博, 二木 厚吉: 代数仕様言語 CafeOBJ による鉄道信号システムの記述と検証. 情報処理学会 第 10 回システム評価研究会. 1999.

[10] 清野 貴博, 二木 厚吉: 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 – 鉄道信号システムの形式仕様とその検証 –. 情報処理学会第 57 回 (平成 10 年度後期) 全国大会, Vol. 1, pp. 236–237. 1998.

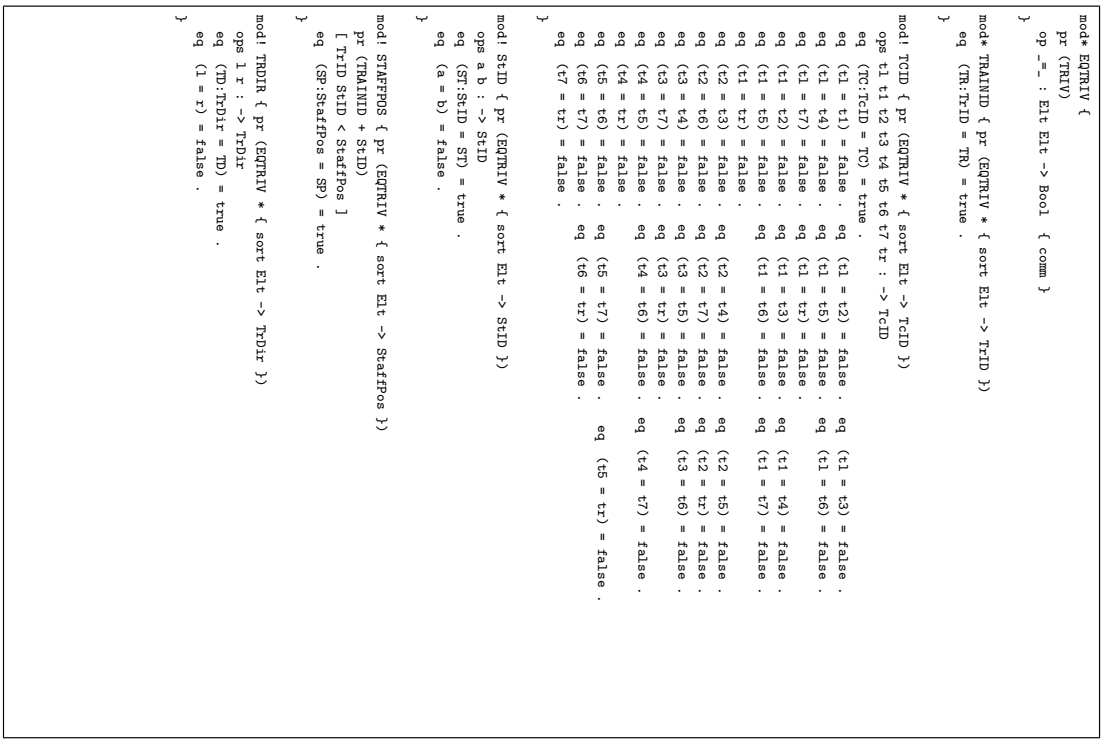
第 A 章

スタフ閉そく

第 3 章および第 4 章で述べた，スタフ閉そくの仕様とその安全性検証に用いた完全なコードを以下に示す.

A.1　仕様記述

A.1.1　スタフ閉そくで使用するデータ



```
mod* EQTRIV {
    pr (TRIV)
    op _=_ : Elt Elt -> Bool  { comm }
}

mod* TRAINID { pr (EQTRIV * { sort Elt -> TrID }) 
    eq  (TR:TrID = TR) = true .
}

mod! TCID { pr (EQTRIV * { sort Elt -> TcID }) 
    ops t1 t1 t2 t3 t4 t5 t6 t7 tr : -> TcID
    eq  (TC:TcID = TC) = true .
    eq  (t1 = t1) = false .  eq  (t1 = t2) = false .  eq  (t1 = t3) = false .
    eq  (t1 = t4) = false .  eq  (t1 = t5) = false .  eq  (t1 = t6) = false .
    eq  (t1 = t7) = false .  eq  (t1 = tr) = false .
    eq  (t1 = t2) = false .  eq  (t1 = t3) = false .  eq  (t1 = t4) = false .
    eq  (t1 = t5) = false .  eq  (t1 = t6) = false .  eq  (t1 = t7) = false .
    eq  (t1 = tr) = false .
    eq  (t2 = t3) = false .  eq  (t2 = t4) = false .  eq  (t2 = t5) = false .
    eq  (t2 = t6) = false .  eq  (t2 = t7) = false .  eq  (t2 = tr) = false .
    eq  (t3 = t4) = false .  eq  (t3 = t5) = false .  eq  (t3 = t6) = false .
    eq  (t3 = t7) = false .  eq  (t3 = tr) = false .
    eq  (t4 = t5) = false .  eq  (t4 = t6) = false .  eq  (t4 = t7) = false .
    eq  (t4 = tr) = false .
    eq  (t5 = t6) = false .  eq  (t5 = t7) = false .   eq  (t5 = tr) = false .
    eq  (t6 = t7) = false .  eq  (t6 = tr) = false .
    eq  (t7 = tr) = false .
}

mod! StID { pr (EQTRIV * { sort Elt -> StID }) 
    ops a b : -> StID
    eq  (ST:StID = ST) = true .
    eq  (a = b) = false .
}

mod! STAFFPOS { pr (EQTRIV * { sort Elt -> StaffPos }) 
    pr (TRAINID + StID)
    [ TrID StID < StaffPos ]
    eq  (SP:StaffPos = SP) = true .
}

mod! TRDIR { pr (EQTRIV * { sort Elt -> TrDir }) 
    ops l r : -> TrDir
    eq  (TD:TrDir = TD) = true .
    eq  (l = r) = false .
}
```

A.1.2 スタフ閉そくを表す状態機械

```
mod* STAFFSYSTEM {
  pr (TCID + TRDIR + STAFFPOS)
  *[ State ]*

  op  init  : -> State                    -- initial state.

  bop pos : TrID State -> TcID            -- observation.
  bop dir : TrID State -> TrDir           -- observation.
  bop staff : State -> StaffPos           -- observation.

  bop move : TrID TcID TrDir State -> State  -- action.
  bop catch : StID TrID State -> State       -- action.
  bop release : StID TrID State -> State     -- action.

  vars TC TC' : TcID
  vars TR TR' : TrID
  vars ST ST' : StID
  var  S : State
  var  TD : TrDir

  -- ------------------------------------------------------------

  ceq pos (TR, init) = tl  if  dir (TR, init) = r .
  ceq pos (TR, init) = tr  if  dir (TR, init) = l .
  eq  staff (init) = a .

  -- ------------------------------------------------------------

  op  c-move : TrID TcID TrDir State -> Bool
  op  e-pos-of-move : TrID TrID TcID TrDir State -> TcID

  eq  c-move (TR', tl, r, S) = pos (TR', S) = tl  and  dir (TR', S) = r .
  ceq e-pos-of-move (TR, TR', tl, r, S) = tl  if  TR = TR' .
  ceq e-pos-of-move (TR, TR', tl, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', tl, l, S):
  eq  c-move (TR', tl, l, S) = false .

  -- for move (TR', t1, r, S):
  eq  c-move (TR', t1, r, S) = pos (TR', S) = t1 and dir (TR', S) = r
                           and staff (S) = TR' .
  ceq e-pos-of-move (TR, TR', t1, r, S) = t3      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t1, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t1, l, S):
  eq  c-move (TR', t1, l, S) = false .

  -- for move (TR', t2, r, S):
  eq  c-move (TR', t2, r, S) = false .

  -- for move (TR', t2, l, S):
  eq  c-move (TR', t2, l, S) = pos (TR', S) = t2 and dir (TR', S) = l
                           and not (staff (S) = TR') .
  ceq e-pos-of-move (TR, TR', t2, l, S) = tl      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t2, l, S) = pos (TR, S)  if  not (TR = TR') .

  ceq e-pos-of-move (TR, TR', t2, l, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t3, r, S):
  eq  c-move (TR', t3, r, S) = pos (TR', S) = t3  and  dir (TR', S) = r .
  ceq e-pos-of-move (TR, TR', t3, r, S) = t4      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t3, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t3, l, S):
  eq  c-move (TR', t3, l, S) = pos (TR', S) = t3  and  dir (TR', S) = l .
  ceq e-pos-of-move (TR, TR', t3, l, S) = t2      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t3, l, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t4, r, S):
  eq  c-move (TR', t4, r, S) = pos (TR', S) = t4  and  dir (TR', S) = r .
  ceq e-pos-of-move (TR, TR', t4, r, S) = t5      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t4, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t4, l, S):
  eq  c-move (TR', t4, l, S) = pos (TR', S) = t4  and  dir (TR', S) = l .
  ceq e-pos-of-move (TR, TR', t4, l, S) = t3      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t4, l, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t5, r, S):
  eq  c-move (TR', t5, r, S) = pos (TR', S) = t5  and  dir (TR', S) = r .
  ceq e-pos-of-move (TR, TR', t5, r, S) = t6      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t5, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t5, l, S):
  eq  c-move (TR', t5, l, S) = pos (TR', S) = t5  and  dir (TR', S) = l .
  ceq e-pos-of-move (TR, TR', t5, l, S) = t4      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t5, l, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t6, r, S):
  eq  c-move (TR', t6, r, S) = pos (TR', S) = t6 and dir (TR', S) = r
                           and not (staff (S) = TR') .
  ceq e-pos-of-move (TR, TR', t6, r, S) = tr      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t6, r, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', t6, l, S):
  eq  c-move (TR', t6, l, S) = false .

  -- for move (TR', t7, r, S):
  eq  c-move (TR', t7, r, S) = false .

  -- for move (TR', t7, l, S):
  eq  c-move (TR', t7, l, S) = pos (TR', S) = t7 and dir (TR', S) = l
                           and staff (S) = TR' .
  ceq e-pos-of-move (TR, TR', t7, l, S) = t5      if  TR = TR' .
  ceq e-pos-of-move (TR, TR', t7, l, S) = pos (TR, S)  if  not (TR = TR') .

  -- for move (TR', tr, r, S):
  eq  c-move (TR', tr, r, S) = false .
```

```
-- for move (TR', tr, l, S):
eq  c-move (TR', tr, l, S) = pos (TR', S) = tr  and  dir (TR', S) = l .
ceq e-pos-of-move (TR, TR', tr, l, S) = t7           if  TR = TR' .
ceq e-pos-of-move (TR, TR', tr, l, S) = pos (TR, S)  if  not (TR = TR') .

-- generic behavior for move (TR', TC', TD, S).
ceq pos (TR, move (TR', TC', TD, S)) = e-pos-of-move (TR, TR', TC', TD, S)
    if  c-move (TR', TC', TD, S) .
ceq pos (TR, move (TR', TC', TD, S)) = pos (TR, S)
    if  not (c-move (TR', TC', TD, S)) .
eq  dir (TR, move (TR', TC', TD, S)) = dir (TR, S) .
eq  staff (move (TR', TC', TD, S)) = staff (S) .

-- ---------------------------------------------------------------------

op  c-catch : StID TrID State -> Bool
op  e-staff-of-catch : StID TrID State -> StaffPos

-- for catch (a, TR', S):
eq  c-catch (a, TR', S) = pos (TR', S) = t1  and  staff (S) = a .
eq e-staff-of-catch (a, TR', S) = TR' .

-- for catch (b, TR', S):
eq  c-catch (b, TR', S) = pos (TR', S) = t7  and  staff (S) = b .
eq e-staff-of-catch (b, TR', S) = TR' .

-- generic behavior for catch (ST, TR', S).
eq  pos (TR, catch (ST, TR', S)) = pos (TR, S) .
eq  dir (TR, catch (ST, TR', S)) = dir (TR, S) .
ceq staff (catch (ST, TR', S)) = e-staff-of-catch (ST, TR', S)
    if  c-catch (ST, TR', S) .
ceq staff (catch (ST, TR', S)) = staff (S)
    if  not (c-catch (ST, TR', S)) .

-- ---------------------------------------------------------------------

op  c-release : StID TrID State -> Bool
op  e-staff-of-release : StID TrID State -> StaffPos

-- for release (a, TR', S):
eq  c-release (a, TR', S) = (pos (TR', S) = t1  or  pos (TR', S) = t2)
                          and  staff (S) = TR' .
eq  e-staff-of-release (a, TR', S) = a .

-- for release (b, TR', S):
eq  c-release (b, TR', S) = (pos (TR', S) = t6  or  pos (TR', S) = t7)
                          and  staff (S) = TR' .
eq  e-staff-of-release (b, TR', S) = b .

-- generic behavior for release (ST, TR', S).
eq  pos (TR, release (ST, TR', S)) = pos (TR, S) .
eq  dir (TR, release (ST, TR', S)) = dir (TR, S) .
ceq staff (release (ST, TR', S)) = e-staff-of-release (ST, TR', S)
    if  c-release (ST, TR', S) .
  ceq staff (release (ST, TR', S)) = staff (S)
    if  not (c-release (ST, TR', S)) .
}
```

A.2 安全性の検証

A.2.1 原始隠蔽定数の定義

```
mod CASES { pr (STAFF)

  *[ Atom < Case < State ]*

  op  s    : -> Case
  op _o_  : Case Case -> Case { coherent assoc comm }
  op comp : Atom Case -> Bool  { coherent }
  op tbl  : Atom -> Case       { coherent }
  op _::_ : Case Case -> Case  { coherent assoc comm }
  op isin : Case Case -> Bool  { coherent }

  op  unreachable : -> Atom

  vars A A' : Atom
  vars C C' : Case
  eq  comp (A, C) = not (isin (C, tbl (A) :: unreachable) == true) .
  eq  isin (A,  A)      = true .
  eq  isin (A o C, A)   = true .
  eq  isin (A o C, A :: C') = true .
  eq  isin (A o C, A :: C') = true .

  -- ---------------------------------------------------------

  -- for any train.
  ops c1 c2 : -> TrID
  eq  (c1 = c2) = false .
  eq  (c1 = a) = false .
  eq  (c1 = b) = false .
  eq  (c2 = a) = false .
  eq  (c2 = b) = false .

  var TC : TcID
  var TD : TrDir
  var SP : StaffPos

  -- ---------------------------------------------------------

  ops  c1t1 c1t1 c1t2 c1t3 c1t4 c1t5 c1t6 c1t7 c1tr : -> Atom
  ops ~c1t1 ~c1t1 ~c1t2 ~c1t3 ~c1t4 ~c1t5 ~c1t6 ~c1t7 ~c1tr : -> Atom

  ceq (pos (c1, ( c1t1 o C)) = TC) = TC = t1   if  comp ( c1t1, C) .
  ceq (pos (c1, (~c1t1 o C)) = t1) = false     if  comp (~c1t1, C) .
  ceq (pos (c1, ( c1t1 o C)) = TC) = TC = t1   if  comp ( c1t1, C) .
  ceq (pos (c1, ( c1t2 o C)) = t1) = false     if  comp ( c1t2, C) .
  ceq (pos (c1, ( c1t2 o C)) = TC) = TC = t2   if  comp ( c1t2, C) .
  ceq (pos (c1, (~c1t2 o C)) = t2) = false     if  comp (~c1t2, C) .
  ceq (pos (c1, ( c1t3 o C)) = TC) = TC = t3   if  comp ( c1t3, C) .
  ceq (pos (c1, (~c1t3 o C)) = t3) = false     if  comp (~c1t3, C) .
  ceq (pos (c1, ( c1t4 o C)) = TC) = TC = t4   if  comp ( c1t4, C) .
  ceq (pos (c1, (~c1t4 o C)) = t4) = false     if  comp (~c1t4, C) .
  ceq (pos (c1, ( c1t5 o C)) = TC) = TC = t5   if  comp ( c1t5, C) .
  ceq (pos (c1, (~c1t5 o C)) = t5) = false     if  comp (~c1t5, C) .
  ceq (pos (c1, ( c1t6 o C)) = TC) = TC = t6   if  comp ( c1t6, C) .
  ceq (pos (c1, (~c1t6 o C)) = t6) = false     if  comp (~c1t6, C) .
  ceq (pos (c1, ( c1t7 o C)) = TC) = TC = t7   if  comp ( c1t7, C) .
  ceq (pos (c1, (~c1t7 o C)) = t7) = false     if  comp (~c1t7, C) .
  ceq (pos (c1, ( c1tr o C)) = TC) = TC = tr   if  comp ( c1tr, C) .
  ceq (pos (c1, (~c1tr o C)) = tr) = false     if  comp (~c1tr, C) .

  eq  tbl ( c1t1) = ~c1t1 :: c1t1 :: c1t2
                  :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t1) = c1t1 .
  eq  tbl ( c1t1) = c1t1 :: ~c1t1 :: c1t2
                  :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t2) = c1t2 .
  eq  tbl ( c1t2) = c1t1 :: c1t2
                  :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t3) = c1t3 .
  eq  tbl ( c1t3) = c1t1 :: c1t2
                  :: c1t3 :: ~c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t4) = c1t4 .
  eq  tbl ( c1t4) = c1t1 :: c1t2 :: c1t3
                  :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t5) = c1t5 .
  eq  tbl ( c1t5) = c1t1 :: c1t2 :: c1t3 :: c1t4
                  :: c1t5 :: ~c1t6 :: c1t7 :: c1tr .
  eq  tbl (~c1t6) = c1t6 .
  eq  tbl ( c1t6) = c1t1 :: c1t2 :: c1t3 :: c1t4 :: c1t5
                  :: c1t6 :: ~c1t7 :: c1tr .
  eq  tbl (~c1t7) = c1t7 .
  eq  tbl ( c1t7) = c1t1 :: c1t2 :: c1t3 :: c1t4 :: c1t5
                  :: c1t6 :: c1t7 :: ~c1tr .
  eq  tbl (~c1tr) = c1tr .

  -- ---------------------------------------------------------

  ops  c2t1 c2t1 c2t2 c2t3 c2t4 c2t5 c2t6 c2t7 c2tr : -> Atom
  ops ~c2t1 ~c2t1 ~c2t2 ~c2t3 ~c2t4 ~c2t5 ~c2t6 ~c2t7 ~c2tr : -> Atom

  ceq (pos (c2, ( c2t1 o C)) = TC) = TC = t1   if  comp ( c2t1, C) .
  ceq (pos (c2, (~c2t1 o C)) = t1) = false     if  comp (~c2t1, C) .
  ceq (pos (c2, ( c2t1 o C)) = TC) = TC = t1   if  comp ( c2t1, C) .
  ceq (pos (c2, ( c2t2 o C)) = t1) = false     if  comp ( c2t2, C) .
  ceq (pos (c2, ( c2t2 o C)) = TC) = TC = t2   if  comp ( c2t2, C) .
  ceq (pos (c2, (~c2t2 o C)) = t2) = false     if  comp (~c2t2, C) .
  ceq (pos (c2, ( c2t3 o C)) = TC) = TC = t3   if  comp ( c2t3, C) .
  ceq (pos (c2, (~c2t3 o C)) = t3) = false     if  comp (~c2t3, C) .
  ceq (pos (c2, ( c2t4 o C)) = TC) = TC = t4   if  comp ( c2t4, C) .
  ceq (pos (c2, (~c2t4 o C)) = t4) = false     if  comp (~c2t4, C) .
  ceq (pos (c2, ( c2t5 o C)) = TC) = TC = t5   if  comp ( c2t5, C) .
  ceq (pos (c2, (~c2t5 o C)) = t5) = false     if  comp (~c2t5, C) .
  ceq (pos (c2, ( c2t6 o C)) = TC) = TC = t6   if  comp ( c2t6, C) .
  ceq (pos (c2, (~c2t6 o C)) = t6) = false     if  comp (~c2t6, C) .
  ceq (pos (c2, ( c2t7 o C)) = TC) = TC = t7   if  comp ( c2t7, C) .
```

```
ceq (pos (c2, (~c2t7 o C)) = t7) = false     if  comp (~c2t7, C) .
ceq (pos (c2, ( c2tr o C)) = TC) = TC = tr   if  comp ( c2tr, C) .
ceq (pos (c2, (~c2tr o C)) = tr) = false     if  comp (~c2tr, C) .

eq  tbl ( c2tl) = ~c2tl ::  c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2tl) =  c2tl .
eq  tbl ( c2t1) =  c2tl :: ~c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t1) =  c2t1 .
eq  tbl ( c2t2) =  c2tl ::  c2t1 :: ~c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t2) =  c2t2 .
eq  tbl ( c2t3) =  c2tl ::  c2t1 ::  c2t2
                :: ~c2t3 ::  c2t4 ::  c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t3) =  c2t3 .
eq  tbl ( c2t4) =  c2tl ::  c2t1 ::  c2t2
                ::  c2t3 :: ~c2t4 ::  c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t4) =  c2t4 .
eq  tbl ( c2t5) =  c2tl ::  c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 :: ~c2t5 ::  c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t5) =  c2t5 .
eq  tbl ( c2t6) =  c2tl ::  c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 :: ~c2t6 ::  c2t7 ::  c2tr .
eq  tbl (~c2t6) =  c2t6 .
eq  tbl ( c2t7) =  c2tl ::  c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 ::  c2t6 :: ~c2t7 ::  c2tr .
eq  tbl (~c2t7) =  c2t7 .
eq  tbl ( c2tr) =  c2tl ::  c2t1 ::  c2t2
                ::  c2t3 ::  c2t4 ::  c2t5 ::  c2t6 ::  c2t7 :: ~c2tr .
eq  tbl (~c2tr) =  c2tr .

-- ------------------------------------------------------------------------

ops  c1r  c1l : -> Atom
ops ~c1r ~c1l : -> Atom

ceq (dir (c1, ( c1r o C)) = TD) = TD = r   if  comp ( c1r, C) .
ceq (dir (c1, (~c1r o C)) = r)  = false    if  comp (~c1r, C) .
ceq (dir (c1, ( c1l o C)) = TD) = TD = l   if  comp ( c1l, C) .
ceq (dir (c1, (~c1l o C)) = l)  = false    if  comp (~c1l, C) .

eq  tbl ( c1r) = ~c1r ::  c1l .
eq  tbl (~c1r) =  c1r ::  c1l .
eq  tbl ( c1l) =  c1r :: ~c1l .
eq  tbl (~c1l) =  c1r ::  c1l .

-- ------------------------------------------------------------------------

ops  c2r  c2l : -> Atom
ops ~c2r ~c2l : -> Atom

ceq (dir (c2, ( c2r o C)) = TD) = TD = r   if  comp ( c2r, C) .
ceq (dir (c2, (~c2r o C)) = r)  = false    if  comp (~c2r, C) .
ceq (dir (c2, ( c2l o C)) = TD) = TD = l   if  comp ( c2l, C) .
ceq (dir (c2, (~c2l o C)) = l)  = false    if  comp (~c2l, C) .

eq  tbl ( c2r) = ~c2r ::  c2l .
eq  tbl (~c2r) =  c2r ::  c2l .
eq  tbl ( c2l) =  c2r :: ~c2l .
eq  tbl (~c2l) =  c2r ::  c2l .

-- ------------------------------------------------------------------------

ops  sc1  sc2  sa  sb : -> Atom
ops ~sc1 ~sc2 ~sa ~sb : -> Atom

ceq (staff ( sc1 o C) = SP) = SP = c1   if  comp ( sc1, C) .
ceq (staff (~sc1 o C) = c1) = false     if  comp (~sc1, C) .
ceq (staff ( sc2 o C) = SP) = SP = c2   if  comp ( sc2, C) .
ceq (staff (~sc2 o C) = c2) = false     if  comp (~sc2, C) .
ceq (staff (  sa o C) = SP) = SP =  a   if  comp (  sa, C) .
ceq (staff ( ~sa o C) = a)  = false     if  comp ( ~sa, C) .
ceq (staff (  sb o C) = SP) = SP =  b   if  comp (  sb, C) .
ceq (staff ( ~sb o C) = b)  = false     if  comp ( ~sb, C) .

eq  tbl ( sc1) = ~sc1 ::  sc2 ::  sa ::  sb .
eq  tbl (~sc1) =  sc1 .
eq  tbl ( sc2) =  sc1 :: ~sc2 ::  sa ::  sb .
eq  tbl (~sc2) =  sc2 .
eq  tbl (  sa) =  sc1 ::  sc2 :: ~sa ::  sb .
eq  tbl ( ~sa) =  sa .
eq  tbl (  sb) =  sc1 ::  sc2 ::  sa :: ~sb .
eq  tbl ( ~sb) =  sb .
}
```

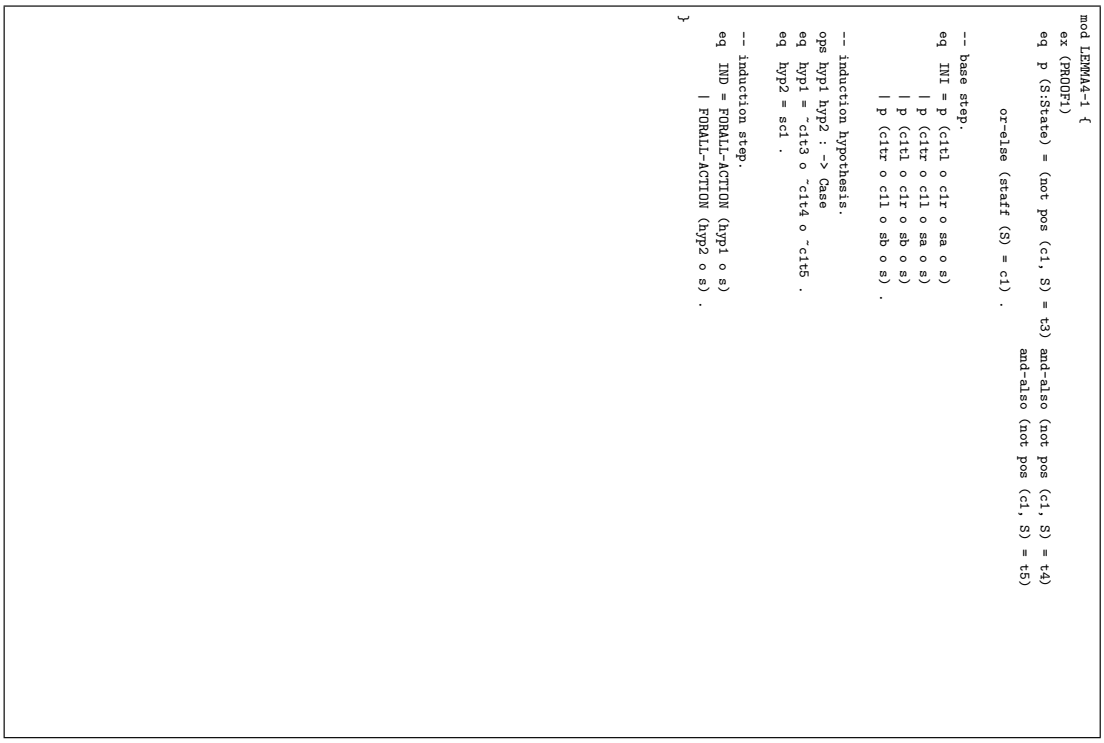
A.2.2 帰納法の検証の定義

```
mod PROOF { ex (CASES) pr (STRING)
   op  p : State -> Bool  -- the predicate to be proven.
   ops INI IND : -> Bool  -- grand terms of the proof.

   op  _|_ : Bool Bool -> Bool { assoc comm idem }

   op  comps : Case Case -> Bool
   eq  comps (A:Atom, C':Case)
             = comp (A, C') .
   eq  comps (A:Atom o A':Atom, C':Case)
             = comp (A, C') and-also comp (A', C') .
   eq  comps (A:Atom o A':Atom o C:Case, C':Case)
             = comp (A, C') and-also comps (A' o C, C') .

-- ==========================================================================
--   tricks for convinience and efficency.
-- ==========================================================================

   op _ => _ : Bool Bool -> Bool { strat: (1 0 2) prec: 61 }
   eq B1:Bool => B2:Bool = (not B1) or-else B2 .

   -- allows to add a simple comment in equation.
   op _ -- _ : Bool String -> Bool { strat: (0) prec: 20 }
   eq B:Bool -- S:String = B .

   -- allows to ingore the boolean expresion with comment.
   op _ --> _ : Bool String -> Bool { strat: (0) prec: 20 }
   eq B:Bool --> S:String = true .

-- ---------------------------------------------------------

   op  MOVE : TrID TcID TrDir Case Case -> Bool
   ceq MOVE (TR:TrID, TC:TcID, TD:TrDir, C:Case, C':Case) = true
      if  comps (C, C') => (p (C o C') => p (move (TR, TC, TD, (C o C')))) .

   op  CATCH : StID TrID Case Case -> Bool
   ceq CATCH (ST:StID, TR:TrID, C:Case, C':Case) = true
      if  comps (C, C') => (p (C o C') => p (catch (ST, TR, (C o C')))) .

   op  RELEASE : StID TrID Case Case -> Bool
   ceq RELEASE (ST:StID, TR:TrID, C:Case, C':Case) = true
      if  comps (C, C') => (p (C o C') => p (release (ST, TR, (C o C')))) .

   op  FORALL-ACTION : Case -> Bool  { memo }

-- ==========================================================================
--   for analysis:
-- ---------------------------------------------------------

   op  traverse : Pairlist Case -> Pairlist
   op  try : Atom Case -> Atom
   eq  traverse (empty, C:Case) = empty .
   eq  traverse (<< A1:Atom, A2:Atom >> :: L:Pairlist, C:Case)
                = << try (A1, C), try (A2, C) >> :: traverse (L, C) .

   bop act : State -> State

   ops tt ff : -> Atom
   ceq try (A:Atom, C:Case) = tt
      if    (comps (A, C) => (p (A o C) => p (act (A o C))) == true) .
   ceq try (A:Atom, C:Case) = ff
      if    (comps (A, C) => (p (A o C) => p (act (A o C))) == false) .
   ceq try (A:Atom, C:Case) = A
      if    (comps (A, C) => (p (A o C) => p (act (A o C))) =/= true)
        and-also (comps (A, C) => (p (A o C) => p (act (A o C))) =/= false) .
}
```

```
-- In an arbitrary train c1:
mod PROOF1 { ex (PROOF)
   eq  FORALL-ACTION (C:Case) =
         MOVE (c1, t1, r, cit1 o cir,          C)
       | MOVE (c1, t1, r, cit1 o cir o sc1,    C)
       | MOVE (c1, t2, r, cit2 o cir,          C)
       | MOVE (c1, t3, r, cit3 o cir,          C)
       | MOVE (c1, t4, r, cit4 o cir,          C)
       | MOVE (c1, t5, r, cit5 o cir,          C)
       | MOVE (c1, t6, r, cit6 o cir o ~sc1,   C)
       | MOVE (c1, t7, r, cit7 o cir,          C)
       | MOVE (c1, tr, r, citr o cir,          C)
       | MOVE (c1, t1, l, cit1 o cil,          C)
       | MOVE (c1, t2, l, cit2 o cil o ~sc1,   C)
       | MOVE (c1, t3, l, cit3 o cil,          C)
       | MOVE (c1, t4, l, cit4 o cil,          C)
       | MOVE (c1, t5, l, cit5 o cil,          C)
       | MOVE (c1, t6, l, cit6 o cil,          C)
       | MOVE (c1, t7, l, cit7 o cil o sc1,    C)
       | MOVE (c1, tr, l, citr o cil,          C)

       | CATCH (a, c1, cit1 o sa,              C)
       | CATCH (b, c1, cit7 o sb,              C)

       | RELEASE (a, c1, cit1 o sc1,           C) | RELEASE (a, c1, cit2 o sc1,
       | RELEASE (b, c1, cit7 o sc1,           C) | RELEASE (b, c1, cit6 o sc1,
                                                                              C) .
}

-- In two arbitrary trains c1 and c2:
mod PROOF2 { ex (PROOF)
   eq  FORALL-ACTION (C:Case) =
         MOVE (c1, t1, r, cit1 o cir,          C)
       | MOVE (c1, t1, r, cit1 o cir o sc1,    C)
       | MOVE (c1, t2, r, cit2 o cir,          C)
       | MOVE (c1, t3, r, cit3 o cir,          C)
       | MOVE (c1, t4, r, cit4 o cir,          C)
       | MOVE (c1, t5, r, cit5 o cir,          C)
       | MOVE (c1, t6, r, cit6 o cir o ~sc1,   C)
```

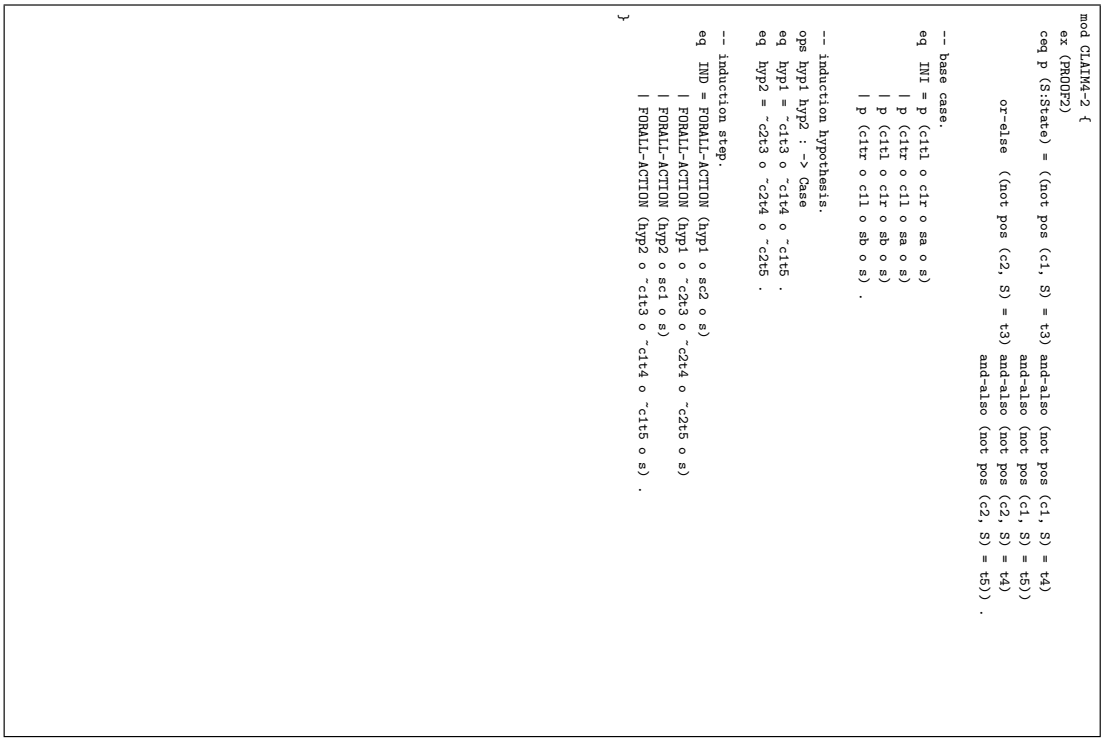
```
  | MOVE (c1, t7, r, c1t7 o c1r,                    c)
  | MOVE (c1, tr, r, c1tr o c1r,                    c)
  | MOVE (c1, tl, l, c1t1 o c1l,                    c)
  | MOVE (c1, t1, l, c1t1 o c1l,                    c)
  | MOVE (c1, t2, l, c1t2 o c1l o ~sc1,             c)
  | MOVE (c1, t3, l, c1t3 o c1l,                    c)
  | MOVE (c1, t4, l, c1t4 o c1l,                    c)
  | MOVE (c1, t5, l, c1t5 o c1l,                    c)
  | MOVE (c1, t6, l, c1t6 o c1l,                    c)
  | MOVE (c1, t7, l, c1t7 o c1l o sc1,              c)
  | MOVE (c1, tr, l, c1tr o c1l,                    c)

  | MOVE (c2, tl, r, c2t1 o c2r,                    c)
  | MOVE (c2, t1, r, c2t1 o c2r o sc2,              c)
  | MOVE (c2, t2, r, c2t2 o c2r,                    c)
  | MOVE (c2, t3, r, c2t3 o c2r,                    c)
  | MOVE (c2, t4, r, c2t4 o c2r,                    c)
  | MOVE (c2, t5, r, c2t5 o c2r,                    c)
  | MOVE (c2, t6, r, c2t6 o c2r o ~sc2,             c)
  | MOVE (c2, t7, r, c2t7 o c2r,                    c)
  | MOVE (c2, tr, r, c2tr o c2r,                    c)
  | MOVE (c2, tl, l, c2t1 o c2l,                    c)
  | MOVE (c2, t1, l, c2t1 o c2l,                    c)
  | MOVE (c2, t2, l, c2t2 o c2l o ~sc2,             c)
  | MOVE (c2, t3, l, c2t3 o c2l,                    c)
  | MOVE (c2, t4, l, c2t4 o c2l,                    c)
  | MOVE (c2, t5, l, c2t5 o c2l,                    c)
  | MOVE (c2, t6, l, c2t6 o c2l,                    c)
  | MOVE (c2, t7, l, c2t7 o c2l o sc2,              c)
  | MOVE (c2, tr, l, c2tr o c2l,                    c)

  | MOVE (c3, tl, r, c3t1 o c3r,                    c)
  | MOVE (c3, t1, r, c3t1 o c3r o sc3,              c)
  | MOVE (c3, t2, r, c3t2 o c3r,                    c)
  | MOVE (c3, t3, r, c3t3 o c3r,                    c)
  | MOVE (c3, t4, r, c3t4 o c3r,                    c)
  | MOVE (c3, t5, r, c3t5 o c3r,                    c)
  | MOVE (c3, t6, r, c3t6 o c3r o ~sc3,             c)
  | MOVE (c3, t7, r, c3t7 o c3r,                    c)
  | MOVE (c3, tr, r, c3tr o c3r,                    c)
  | MOVE (c3, tl, l, c3t1 o c3l,                    c)
  | MOVE (c3, t1, l, c3t1 o c3l,                    c)
  | MOVE (c3, t2, l, c3t2 o c3l o ~sc3,             c)
  | MOVE (c3, t3, l, c3t3 o c3l,                    c)
  | MOVE (c3, t4, l, c3t4 o c3l,                    c)
  | MOVE (c3, t5, l, c3t5 o c3l,                    c)
  | MOVE (c3, t6, l, c3t6 o c3l,                    c)
  | MOVE (c3, t7, l, c3t7 o c3l o sc3,              c)
  | MOVE (c3, tr, l, c3tr o c3l,                    c)

  | CATCH (a, c1, c1t1 o sa,                        c)
  | CATCH (b, c1, c1t7 o sb,                        c)
  | CATCH (a, c2, c2t1 o sa,                        c)
  | CATCH (b, c2, c2t7 o sb,                        c)
  | CATCH (a, c3, c3t1 o sa,                        c)
  | CATCH (b, c3, c3t7 o sb,                        c)

  | RELEASE (a, c1, c1t1 o sc1,                     c) | RELEASE (a, c1, c1t2 o sc1,     c)
  | RELEASE (b, c1, c1t7 o sc1,                     c) | RELEASE (b, c1, c1t6 o sc1,     c)
  | RELEASE (a, c2, c2t1 o sc2,                     c) | RELEASE (a, c2, c2t2 o sc2,     c)
  | RELEASE (b, c2, c2t7 o sc2,                     c) | RELEASE (b, c2, c2t6 o sc2,     c)
  | RELEASE (a, c3, c3t1 o sc3,                     c) | RELEASE (a, c3, c3t2 o sc3,     c)
  | RELEASE (b, c3, c3t7 o sc3,                     c) | RELEASE (b, c3, c3t6 o sc3,     c) .
}
```

A.2.3 補題 4.1



```
mod LEMMA4-1 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t3) and-also (not pos (c1, S) = t4)
                                           and-also (not pos (c1, S) = t5)
                    or-else (staff (S) = c1) .

  -- base step.
  eq  INI = p (c1tl o c1r o sa o s)
            | p (c1tr o c1l o sa o s)
            | p (c1tl o c1r o sb o s)
            | p (c1tr o c1l o sb o s) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t3 o ~c1t4 o ~c1t5 .
  eq  hyp2 = sc1 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o s)
            | FORALL-ACTION (hyp2 o s) .
}
```

A.2.4 表明 4.2



```
mod CLAIM4-2 {
  ex (PROOF2)
  ceq p (S:State) = ((not pos (c1, S) = t3) and-also (not pos (c1, S) = t4)
                                            and-also (not pos (c1, S) = t5))
                    or-else  ((not pos (c2, S) = t3) and-also (not pos (c2, S) = t4)
                                                     and-also (not pos (c2, S) = t5)) .

  -- base case.
  eq  INI = p (c1tl o c1r o sa o s)
            | p (c1tr o c1l o sa o s)
            | p (c1tl o c1r o sb o s)
            | p (c1tr o c1l o sb o s) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t3 o ~c1t4 o ~c1t5 .
  eq  hyp2 = ~c2t3 o ~c2t4 o ~c2t5 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o sc2 o s)
            | FORALL-ACTION (hyp1 o ~c2t3 o ~c2t4 o ~c2t5 o s)
            | FORALL-ACTION (hyp2 o sc1 o s)
            | FORALL-ACTION (hyp2 o ~c1t3 o ~c1t4 o ~c1t5 o s) .
}
```

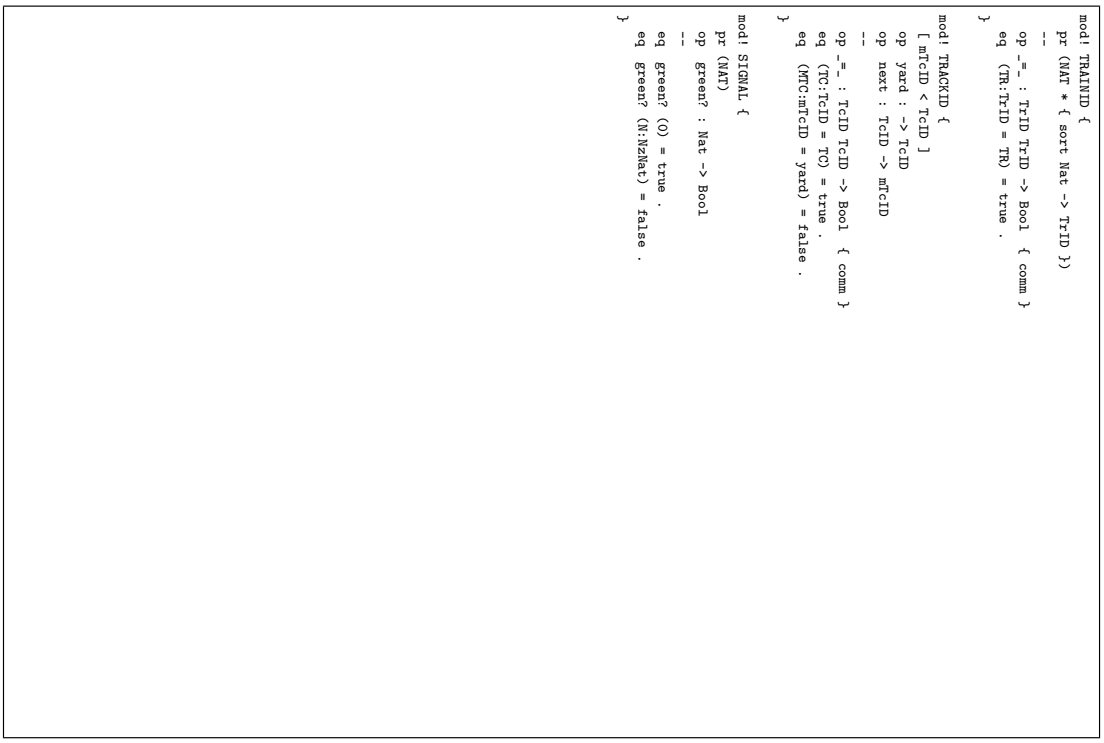
第 B 章

複線用自動閉そく

第 5 章で述べた，複線用自動閉そくの仕様とその安全性検証に用いた完全なコードを以下に示す.

B.1　仕様

B.1.1　複線用自動閉そくで使用するデータ



```
mod! TRAINID {
  pr (NAT * { sort Nat -> TrID })
  --
  op _=_ : TrID TrID -> Bool { comm }
  eq (TR:TrID = TR) = true .
}

mod! TRACKID {
  [ mTcID < TcID ]
  op yard : -> TcID
  op next : TcID -> mTcID
  --
  op _=_ : TcID TcID -> Bool { comm }
  eq (TC:TcID = TC) = true .
  eq (MTC:mTcID = yard) = false .
}

mod! SIGNAL {
  pr (NAT)
  op green? : Nat -> Bool
  --
  eq green? (0) = true .
  eq green? (N:NzNat) = false .
}
```

B.1.2 複線用自動閉そくを表した状態機械

```
mod* ONEWAY-RAILROAD-SYSTEM {
  pr (TRAINID + TRACKID + SIGNAL)

  *[ State ]*

  op  init : -> State                          -- initial state.
  bop move : TrID TcID State -> State          -- action
  bop pos  : TrID State -> TcID                -- observation
  bop num  : mTcID State -> Nat                -- observation

  op  c-move : TrID TcID State -> Bool
  op  e-pos-move : TrID TrID TcID State -> TcID
  op  e-num-move : mTcID TrID TcID State -> Nat

  var  S : State
  vars TR TR' : TrID
  vars TC TC' : TcID
  vars MTC MTC' : mTcID

  -- initial condition:
  eq  pos (TR, init) = yard .
  eq  num (MTC, init) = 0 .

  -- effective condition for move (TR, TC, S).
  eq  c-move (TR, TC, S) = (pos (TR, S) = TC)
                  and  green? (num (next (TC), S)) .

  -- effects for move (TR, TC, S).
  ceq  e-pos-move (TR, TR', TC', S) = next (pos (TR, S))
     if  TR = TR' .
  ceq  e-pos-move (TR, TR', TC', S) = pos (TR, S)
     if  not (TR = TR') .

  ceq  e-num-move (MTC, TR', TC', S) = s (num (MTC, S))
     if  MTC = next (TC') .
  ceq  e-num-move (MTC, TR', TC', S) = p (num (MTC, S))
     if  MTC = TC' .
  ceq  e-num-move (MTC, TR', TC', S) = num (MTC, S)
     if  not (MTC = TC' or  MTC = next (TC')) .

  -- the behavior.
  ceq pos (TR, move (TR', TC', S)) = e-pos-move (TR, TR', TC', S)
     if  c-move (TR', TC', S) .
  ceq pos (TR, move (TR', TC', S)) = pos (TR, S)
     if  not (c-move (TR', TC', S)) .
  ceq  num (MTC, move (TR', TC', S)) = e-num-move (MTC, TR', TC', S)
     if  c-move (TR', TC', S) .
  ceq  num (MTC, move (TR', TC', S)) = num (MTC, S)
     if  not (c-move (TR', TC', S)) .
}
```

B.2 検証

B.2.1 原始隠蔽定数の定義

```
mod CASES { pr (ONEWAY-RAILROAD-SYSTEM)

  *[ Atom < Case < State ]*

  op  s    : -> Case
  op _°_   : Case Case -> Case  { coherent assoc comm }
  op comp  : Atom Case -> Bool  { coherent }
  op tbl   : Atom -> Case       { coherent }
  op _::_  : Case Case -> Case  { coherent assoc comm }
  op isin  : Case Case -> Bool  { coherent }

  op  unreachable : -> Atom

  vars A A' : Atom
  vars C C' : Case
  eq  comp (A, C) = not (isin (C, tbl (A) :: unreachable) == true) .
  eq  isin (A,     A)      = true .
  eq  isin (A ° C, A)      = true .
  eq  isin (A,     A :: C') = true .
  eq  isin (A ° C, A :: C') = true .

  -- ---------------------------------------------------------

  -- for any train:
  ops c1 c2 : -> TrID

  -- for any tracks:
  ops t t1 t2 t1' t2' : -> nTcID
  eq  next (t1) = t1' .
  eq  next (t2) = t2' .

  ops  cit cit1 cit1' : -> Atom
  ops ~cit ~cit1 ~cit1' : -> Atom

  ceq (pos (c1, (  cit ° C)) = t)   = true    if comp ( cit,  C) .
  ceq (pos (c1, ( ~cit ° C)) = t)   = false   if comp (~cit,  C) .
  ceq (pos (c1, ( cit1 ° C)) = t1)  = true    if comp ( cit1, C) .
  ceq (pos (c1, (~cit1 ° C)) = t1)  = false   if comp (~cit1, C) .
  ceq (pos (c1, ( cit1' ° C)) = t1') = true   if comp ( cit1', C) .
  ceq (pos (c1, (~cit1' ° C)) = t1') = false  if comp (~cit1', C) .

  eq  tbl (  cit)  =  cit .
  eq  tbl ( ~cit)  = ~cit .
  eq  tbl ( cit1)  =  cit1 :: cit1' .
  eq  tbl ( cit1') =  cit1' :: cit1 .
  eq  tbl (~cit1') = ~cit1' .

  -- ---------------------------------------------------------

  ops  c2t  c2t2  c2t2' : -> Atom
  ops ~c2t ~c2t2 ~c2t2' : -> Atom

  ceq (pos (c2, ( c2t  ° C)) = t)   = true    if comp ( c2t,  C) .
  ceq (pos (c2, (~c2t  ° C)) = t)   = false   if comp (~c2t,  C) .
  ceq (pos (c2, ( c2t2 ° C)) = t2)  = true    if comp ( c2t2, C) .
  ceq (pos (c2, (~c2t2 ° C)) = t2)  = false   if comp (~c2t2, C) .
  ceq (pos (c2, ( c2t2' ° C)) = t2') = true   if comp ( c2t2', C) .
  ceq (pos (c2, (~c2t2' ° C)) = t2') = false  if comp (~c2t2', C) .

  eq  tbl ( c2t)  =  ~c2t .
  eq  tbl (~c2t)  =   c2t .
  eq  tbl ( c2t2)  =  ~c2t2 :: c2t2' .
  eq  tbl (~c2t2)  =  c2t2 .
  eq  tbl ( c2t2') =  ~c2t2' :: c2t2 .
  eq  tbl (~c2t2') =  c2t2' .

  -- ---------------------------------------------------------

  ops  gt  gt1  gt1' : -> Atom
  ops ~gt ~gt1 ~gt1' : -> Atom

  ceq (green? (num (t,  gt ° C))) = true   if comp ( gt,  C) .
  ceq (green? (num (t, ~gt ° C))) = false  if comp (~gt,  C) .

  eq  tbl ( gt) = ~gt .

  ceq (green? (num (t1,  gt1 ° C))) = true   if comp ( gt1,  C) .
  ceq (green? (num (t1, ~gt1 ° C))) = false  if comp (~gt1,  C) .
  ceq (green? (num (t1', gt1' ° C))) = true  if comp ( gt1', C) .
  ceq (green? (num (t1', ~gt1' ° C))) = false  if comp (~gt1', C) .

  eq  tbl ( gt1)  =  ~gt1 .
  eq  tbl (~gt1)  =  gt1 .
  eq  tbl ( gt1') =  ~gt1' .
  eq  tbl (~gt1') =  gt1' .

  -- ---------------------------------------------------------

  ops  gt2  gt2' : -> Atom
  ops ~gt2 ~gt2' : -> Atom

  ceq (green? (num (t2,  gt2 ° C))) = true   if comp ( gt2,  C) .
  ceq (green? (num (t2, ~gt2 ° C))) = false  if comp (~gt2,  C) .
  ceq (green? (num (t2', gt2' ° C))) = true  if comp ( gt2', C) .
  ceq (green? (num (t2', ~gt2' ° C))) = false  if comp (~gt2', C) .

  eq  tbl ( gt2)  =  ~gt2 .
  eq  tbl (~gt2)  =  gt2 .
  eq  tbl ( gt2') =  ~gt2' .
  eq  tbl (~gt2') =  gt2' .
}
```

B.2.2 帰納法の検証の定義

```
mod PROOF { ex (CASES) pr (STRING)
  op  p : State -> Bool  -- the predicate to be proven.
  ops IND INI : -> Bool  -- grand terms of the proof.

  op  _|_ : Bool Bool -> Bool { assoc comm idem }

  op  comps : Case Case -> Bool
  eq  comps (A:Atom o A':Atom, C':Case)
            = comp (A, C') and-also comp (A', C') .
  eq  comps (A:Atom o A':Atom o C:Case, C':Case)
            = comp (A, C') and-also comps (A' o C, C') .

-- ---------------------------------------------------------------

  op _ => _ : Bool Bool -> Bool { strat: (1 0 2) prec: 61 }
  eq B1:Bool => B2:Bool = (not B1) or-else B2 .

  -- allows to add a simple comment in equation.
  op _ -- _ : Bool String -> Bool { strat: (0) prec: 20 }
  eq B:Bool --  S:String = B .

  -- allows to ingore the boolean expresion with comment.
  op _ --> _ : Bool String -> Bool { strat: (0) prec: 20 }
  eq B:Bool --> S:String = true .

-- ---------------------------------------------------------------

  op  MOVE  : TrID TcID Case Case -> Bool
  ceq MOVE (TR:TrID, TC:TcID, C:Case, C':Case) = true
    if  comps (C, C') => (p (C o C') => p (move (TR, TC, (C o C')))) .

  op  FORALL-ACTION : Case -> Bool  { memo }
}

mod PROOF1 { ex (PROOF)
  eq  FORALL-ACTION (HYP:Case)
        = MOVE (c1, t1, c1t1 o gt1', HYP) .
}

mod PROOF2 { ex (PROOF)
  eq  FORALL-ACTION (HYP:Case)
        = MOVE (c1, t1, c1t1 o gt1', HYP)
        | MOVE (c2, t2, c2t2 o gt2', HYP) .
}
```

B.2.3 補題 5.1

```
mod LEMMA5-1 {
        ex (PROOF1)
  eq p (S:State) = not (pos (c1, S) = t)
                   or-else  not (green? (num (t, S))) .

  -- base step.
  eq  INI = p (init) .

  -- cases for induction hypothesis.
  ops hyp1 hyp2 hyp3 hyp4 : -> Case
  eq  hyp1 = ~c1t .
  eq  hyp2 = ~gt .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

open LEMMA5-1
eq (t = t1) = true .
eq (t = t1') = false .
eq next(pos(c1,c1t1 o C:Case) = t1' .
red INI | IND .
close

open LEMMA5-1
eq (t = t1') = true .
red INI | IND .
close

open LEMMA5-1
eq (t = t1) = false .
eq (t = t1') = false .
eq next(pos(c1,c1t1 o C:Case)) = t1' .
red INI | IND .
close
```

B.2.4 表明 5.1

```
mod CLAIM5-1 {
    ex (PROOF2)
    eq p (S:State) = (c1 = c2)
        or-else (not (pos (c1, S) = t))
        or-else (not (pos (c2, S) = t)) .

    -- base step.
    eq  INI = p (init) .

    -- cases for induction hypothesis.
    ops hyp2 hyp3 : -> Case
    eq  hyp2 = ~c1t .
    eq  hyp3 = ~c2t .

    ops t1'eqt ~t1'eqt : -> Atom
    ops t2'eqt ~t2'eqt : -> Atom

    var  C : Case

    ceq (next (pos (c1, ( t1'eqt o C))) = t) = true  if  comp ( t1'eqt, C) .
    ceq (next (pos (c1, (~t1'eqt o C))) = t) = false  if  comp (~t1'eqt, C) .
    ceq (next (pos (c2, ( t2'eqt o C))) = t) = true  if  comp ( t2'eqt, C) .
    ceq (next (pos (c2, (~t2'eqt o C))) = t) = false  if  comp (~t2'eqt, C) .

    eq  tbl ( t1'eqt) = ~t1'eqt .
    eq  tbl (~t1'eqt) =  t1'eqt .
    eq  tbl ( t2'eqt) = ~t2'eqt .
    eq  tbl (~t2'eqt) =  t2'eqt .

    -- induction step.
    eq  IND = FORALL-ACTION (hyp2 o  c2t o  t1'eqt o ~gt1')
              | FORALL-ACTION (hyp2 o  c2t o ~t1'eqt)
              | FORALL-ACTION (hyp2 o ~c2t)
              | FORALL-ACTION (hyp3 o  c1t o  t2'eqt o ~gt2')
              | FORALL-ACTION (hyp3 o  c1t o ~t2'eqt)
              | FORALL-ACTION (hyp3 o ~c1t) .
}

open CLAIM5-1
eq (c1 = c2) = false .
red INI | IND .
close

open CLAIM5-1
eq (c1 = c2) = true .
red INI | IND .
close
```

第 C 章

単線用自動閉そく

第 6 章で述べた，単線用自動閉そくの仕様とその安全性検証に用いた完全なコードを以下に示す.

C.1 仕様

C.1.1 単線用自動閉そくで使用するデータ

```
mod! TRAIND {
  pr (NAT * { sort Nat -> TrID })
  op _=_ : TrID TrID -> Bool  { comm }
  eq  (TR:TrID = TR) = true .
}

mod! TCID {
  [ TcID ]
  ops t1 t1 t2 t3 t4 t5 t6 t7 tr : -> TcID
  op _=_ : TcID TcID -> Bool  { comm }
  eq  (TC:TcID = TC) = true .
  eq  (t1 = t1) = false .  eq  (t1 = t2) = false .  eq  (t1 = t3) = false .
  eq  (t1 = t4) = false .  eq  (t1 = t5) = false .  eq  (t1 = t6) = false .
  eq  (t1 = t7) = false .  eq  (t1 = tr) = false .
  eq  (t1 = t2) = false .  eq  (t1 = t3) = false .  eq  (t1 = t4) = false .
  eq  (t1 = t5) = false .  eq  (t1 = t6) = false .  eq  (t1 = t7) = false .
  eq  (t1 = tr) = false .
  eq  (t2 = t3) = false .  eq  (t2 = t4) = false .  eq  (t2 = t5) = false .
  eq  (t2 = t6) = false .  eq  (t2 = t7) = false .  eq  (t2 = tr) = false .
  eq  (t3 = t4) = false .  eq  (t3 = t5) = false .  eq  (t3 = t6) = false .
  eq  (t3 = t7) = false .  eq  (t3 = tr) = false .
  eq  (t4 = t5) = false .  eq  (t4 = t6) = false .  eq  (t4 = t7) = false .
  eq  (t4 = tr) = false .
  eq  (t5 = t6) = false .  eq  (t5 = t7) = false .  eq  (t5 = tr) = false .
  eq  (t6 = t7) = false .  eq  (t6 = tr) = false .
  eq  (t7 = tr) = false .
}

mod! SIGNALID {
  [ SgID ]
  ops s1 s2 s3 s4 : -> SgID
  op _=_ : SgID SgID -> Bool  { comm }
  eq  (SG:SgID = SG) = true .
  eq  (s1 = s2) = false .  eq  (s1 = s3) = false .  eq  (s1 = s4) = false .
  eq  (s2 = s3) = false .  eq  (s2 = s4) = false .
  eq  (s3 = s4) = false .
}

mod! LEVERID {
  [ LvID ]
  ops x y : -> LvID
  op _=_ : LvID LvID -> Bool  { comm }
  eq  (LV:LvID = LV) = true .
  eq  (x = y) = false .
}

mod! TRAINDIR {
  [ TrDir ]
  ops l r   : -> TrDir
  op _=_ : TrDir TrDir -> Bool  { comm }
  eq  (TD:TrDir = TD) = true .
  eq  (l = r) = false .
}
```

```
mod! LEVERDIR {
    [ LvDir ]
    ops l n r : -> LvDir
    op  _=_ : LvDir LvDir -> Bool  { comm }
    eq  (LD:LvDir = LD) = true .
    eq  (l = n) = false .  eq  (l = r) = false .  eq  (n = r) = false .
}
```

C.1.2 単線用自動閉そくを表した状態機械

```
mod* ABB {
  pr (TRAINID + TCID + SIGNALID + LEVERID + TRAINDIR + LEVERDIR + NAT)
  *[ State ]*

    op  init   : -> State                    -- initial state.

    bop signal : SgID State -> Bool          -- observation.
    bop num    : TcID State -> Nat           -- observation.
    bop pos    : TrID State -> TcID          -- observation.
    bop dir    : TrID State -> TrDir         -- observation.
    bop which  : LvID State -> LvDir         -- observation.
    bop using  : SgID State -> Bool          -- observation.
    bop move   : TrID TcID TrDir State -> State  -- action.
    bop turn   : LvID LvDir State -> State   -- action.
    bop allow  : SgID State -> State         -- action.
    bop forbid : SgID State -> State         -- action.

    -- variables.
    vars SG SG' : SgID
    vars TC TC' : TcID
    vars TR TR' : TrID
    vars LV LV' : LvID
    var  S : State
    var  TD : TrDir
    var  LD : LvDir
    var  B : Bool

-- ------------------------------------------------------------

    -- equations for initial state.
    eq  signal (SG, init) = false .
    eq  num (TC, init) = 0 .
    ceq pos (TR, init) = t1  if  dir (TR, init) = r .
    ceq pos (TR, init) = tr  if  dir (TR, init) = l .
    eq  which (LV, init) = n .
    eq  using (SG, init) = false .

    -- equations for actions (transitions).
    op  c-move     :   TrID TcID TrDir State -> Bool
    op  e-signal-move : SgID TrID TcID TrDir State -> Bool
    op  e-num-move    : TcID TrID TcID TrDir State -> Nat
    op  e-pos-move    : TrID TrID TcID TrDir State -> TcID

    -- for move (TR', t1, r, S):
    eq  c-move (TR', t1, r, S) = pos (TR', S) = t1
                             and dir (TR', S) = r
    eq  e-signal-move (SG, TR', t1, r, S) = signal (SG, S) .
                             and not (num (t1, S) > 0) .
    eq  e-num-move (TC, TR', t1, r, S) = s (num (TC, S))  if     TC = t1 .
    ceq e-num-move (TC, TR', t1, r, S) = num (TC, S)      if  not (TC = t1) .
    ceq e-pos-move (TR, TR', t1, r, S) = t1               if     TR = TR' .
    ceq e-pos-move (TR, TR', t1, r, S) = pos (TR, S)      if  not (TR = TR') .

    -- for move (TR', t1, l, S):
    eq  c-move (TR', t1, l, S) = false .

    -- for move (TR', t1, r, S):
    eq  c-move (TR', t1, r, S) = pos (TR', S) = t1
                             and dir (TR', S) = r
                                 and signal (s1, S) .
    ceq e-signal-move (SG, TR', t1, r, S) = false                if     SG = s1 .
    ceq e-signal-move (SG, TR', t1, r, S) = signal (SG, S)       if  not (SG = s1) .
    ceq e-num-move (TC, TR', t1, r, S) = s (num (TC, S))         if     TC = t3 .
    ceq e-num-move (TC, TR', t1, r, S) = p (num (TC, S))         if     TC = t1 .
    ceq e-num-move (TC, TR', t1, r, S) = num (TC, S)             if  not (TC = t1)
                                                                    or  TC = t3) .
    ceq e-pos-move (TR, TR', t1, r, S) = t3                      if     TR = TR' .
    ceq e-pos-move (TR, TR', t1, r, S) = pos (TR, S)             if  not (TR = TR') .

    -- for move (TR', t2, r, S):
    eq  c-move (TR', t2, r, S) = false .

    -- for move (TR', t2, l, S):
    eq  c-move (TR', t2, l, S) = pos (TR', S) = t2
                             and dir (TR', S) = l .
    eq  e-signal-move (SG, TR', t2, l, S) = signal (SG, S) .
    ceq e-num-move (TC, TR', t2, l, S) = p (num (TC, S))         if     TC = t2 .
    ceq e-num-move (TC, TR', t2, l, S) = num (TC, S)             if  not (TC = t2) .
    ceq e-pos-move (TR, TR', t2, l, S) = t1                      if     TR = TR' .
    ceq e-pos-move (TR, TR', t2, l, S) = pos (TR, S)             if  not (TR = TR') .

    -- for move (TR', t3, r, S):
    eq  c-move (TR', t3, r, S) = pos (TR', S) = t3
                             and dir (TR', S) = r .
    eq  e-signal-move (SG, TR', t3, r, S) = signal (SG, S) .
    ceq e-num-move (TC, TR', t3, r, S) = s (num (TC, S))         if     TC = t4 .
    ceq e-num-move (TC, TR', t3, r, S) = p (num (TC, S))         if     TC = t3 .
    ceq e-num-move (TC, TR', t3, r, S) = num (TC, S)             if  not (TC = t3)
                                                                    or  TC = t4) .
    ceq e-pos-move (TR, TR', t3, r, S) = pos (TR, S)             if  not (TR = TR') .

    -- for move (TR', t3, l, S):
    eq  c-move (TR', t3, l, S) = pos (TR', S) = t3
                             and dir (TR', S) = l .
    eq  e-signal-move (SG, TR', t3, l, S) = signal (SG, S) .
    ceq e-num-move (TC, TR', t3, l, S) = s (num (TC, S))         if     TC = t2 .
    ceq e-num-move (TC, TR', t3, l, S) = p (num (TC, S))         if     TC = t3 .
    ceq e-num-move (TC, TR', t3, l, S) = num (TC, S)             if  not (TC = t2)
                                                                    or  TC = t3) .
    ceq e-pos-move (TR, TR', t3, l, S) = t2                      if     TR = TR' .
    ceq e-pos-move (TR, TR', t3, l, S) = pos (TR, S)             if  not (TR = TR') .

    -- for move (TR', t4, r, S):
```

```
eq  c-move (TR', t4, r, S) = pos (TR', S) = t4
                       and  dir (TR', S) = r
                       and  signal (s3, S) .
ceq e-signal-move (SG, TR', t4, r, S) = false          if       SG = s3 .
ceq e-signal-move (SG, TR', t4, r, S) = signal (SG, S)  if  not (SG = s3) .
ceq e-num-move (TC, TR', t4, r, S) = s (num (TC, S))    if       TC = t5 .
ceq e-num-move (TC, TR', t4, r, S) = p (num (TC, S))    if       TC = t4 .
ceq e-num-move (TC, TR', t4, r, S) = num (TC, S)        if  not (TC = t4
                                                             or  TC = t5) .
ceq e-pos-move (TR, TR', t4, r, S) = t5                 if       TR = TR' .
ceq e-pos-move (TR, TR', t4, r, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', t4, l, S):
eq  c-move (TR', t4, l, S) = pos (TR', S) = t4
                       and  dir (TR', S) = l
                       and  signal (s2, S) .
ceq e-signal-move (SG, TR', t4, l, S) = false          if       SG = s2 .
ceq e-signal-move (SG, TR', t4, l, S) = signal (SG, S)  if  not (SG = s2) .
ceq e-num-move (TC, TR', t4, l, S) = s (num (TC, S))    if       TC = t3 .
ceq e-num-move (TC, TR', t4, l, S) = p (num (TC, S))    if       TC = t4 .
ceq e-num-move (TC, TR', t4, l, S) = num (TC, S)        if  not (TC = t3
                                                             or  TC = t4) .
ceq e-pos-move (TR, TR', t4, l, S) = t3                 if       TR = TR' .
ceq e-pos-move (TR, TR', t4, l, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', t5, r, S):
eq  c-move (TR', t5, r, S) = pos (TR', S) = t5
                       and  dir (TR', S) = r .
eq  e-signal-move (SG, TR', t5, r, S) = signal (SG, S) .
ceq e-num-move (TC, TR', t5, r, S) = s (num (TC, S))    if       TC = t6 .
ceq e-num-move (TC, TR', t5, r, S) = p (num (TC, S))    if       TC = t5 .
ceq e-num-move (TC, TR', t5, r, S) = num (TC, S)        if  not (TC = t5
                                                             or  TC = t6) .
ceq e-pos-move (TR, TR', t5, r, S) = t6                 if       TR = TR' .
ceq e-pos-move (TR, TR', t5, r, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', t5, l, S):
eq  c-move (TR', t5, l, S) = pos (TR', S) = t5
                       and  dir (TR', S) = l .
eq  e-signal-move (SG, TR', t5, l, S) = signal (SG, S) .
ceq e-num-move (TC, TR', t5, l, S) = s (num (TC, S))    if       TC = t4 .
ceq e-num-move (TC, TR', t5, l, S) = p (num (TC, S))    if       TC = t5 .
ceq e-num-move (TC, TR', t5, l, S) = num (TC, S)        if  not (TC = t4
                                                             or  TC = t5) .
ceq e-pos-move (TR, TR', t5, l, S) = t4                 if       TR = TR' .
ceq e-pos-move (TR, TR', t5, l, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', t6, r, S):
eq  c-move (TR', t6, r, S) = pos (TR', S) = t6
                       and  dir (TR', S) = r .
eq  e-signal-move (SG, TR', t6, r, S) = signal (SG, S) .
ceq e-num-move (TC, TR', t6, r, S) = p (num (TC, S))    if       TC = t6 .
ceq e-num-move (TC, TR', t6, r, S) = num (TC, S)        if  not (TC = t6) .
```

```
ceq e-pos-move (TR, TR', t6, r, S) = tr                 if       TR = TR' .
ceq e-pos-move (TR, TR', t6, r, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', t6, l, S):
eq  c-move (TR', t6, l, S) = false .

-- for move (TR', t7, r, S):
eq  c-move (TR', t7, r, S) = false .

-- for move (TR', t7, l, S):
eq  c-move (TR', t7, l, S) = pos (TR', S) = t7
                       and  dir (TR', S) = l
                       and  signal (s4, S) .
ceq e-signal-move (SG, TR', t7, l, S) = false          if       SG = s4 .
ceq e-signal-move (SG, TR', t7, l, S) = signal (SG, S)  if  not (SG = s4) .
ceq e-num-move (TC, TR', t7, l, S) = s (num (TC, S))    if       TC = t5 .
ceq e-num-move (TC, TR', t7, l, S) = p (num (TC, S))    if       TC = t7 .
ceq e-num-move (TC, TR', t7, l, S) = num (TC, S)        if  not (TC = t5
                                                             or  TC = t7) .
ceq e-pos-move (TR, TR', t7, l, S) = t5                 if       TR = TR' .
ceq e-pos-move (TR, TR', t7, l, S) = pos (TR, S)        if  not (TR = TR') .

-- for move (TR', tr, r, S):
eq  c-move (TR', tr, r, S) = false .

-- for move (TR', tr, l, S):
eq  c-move (TR', tr, l, S) = pos (TR', S) = tr
                       and  dir (TR', S) = l
                       and  not (num (t7, S) > 0) .
eq  e-signal-move (SG, TR', tr, l, S) = signal (SG, S) .
ceq e-num-move (TC, TR', tr, l, S) = s (num (TC, S))    if       TC = t7 .
ceq e-num-move (TC, TR', tr, l, S) = num (TC, S)        if  not (TC = t7) .
ceq e-pos-move (TR, TR', tr, l, S) = t7                 if       TR = TR' .
ceq e-pos-move (TR, TR', tr, l, S) = pos (TR, S)        if  not (TR = TR') .

-- generic behavior for move (TR', TC', TD, S).
ceq signal (SG, move (TR', TC', TD, S)) = e-signal-move (SG, TR', TC', TD, S)
    if  c-move (TR', TC', TD, S) .
ceq signal (SG, move (TR', TC', TD, S)) = signal (SG, S)
    if  not (c-move (TR', TC', TD, S)) .
ceq num (TC, move (TR', TC', TD, S)) = e-num-move (TC, TR', TC', TD, S)
    if  c-move (TR', TC', TD, S) .
ceq num (TC, move (TR', TC', TD, S)) = num (TC, S)
    if  not (c-move (TR', TC', TD, S)) .
ceq pos (TR, move (TR', TC', TD, S)) = e-pos-move (TR, TR', TC', TD, S)
    if  c-move (TR', TC', TD, S) .
ceq pos (TR, move (TR', TC', TD, S)) = pos (TR, S)
    if  not (c-move (TR', TC', TD, S)) .
eq  dir (TR, move (TR', TC, TD, S)) = dir (TR, S) .
eq  which (LV, move (TR', TC, TD, S)) = which (LV, S) .
eq  using (SG, move (TR', TC, TD, S)) = using (SG, S) .

-- ---------------------------------------------------------------------
```

```
-- ------------------------------------------------------------------------

  op  c-turn       :       LvID LvDir State -> Bool
  op  e-which-turn : LvID  LvID LvDir State -> LvDir

  -- for turn (x, l, S):
  eq  c-turn (x, l, S) = which (x, S) = n
                     and  which (y, S) = n .
  ceq e-which-turn (LV, x, l, S) = l                 if       LV = x .
  ceq e-which-turn (LV, x, l, S) = which (LV, S)     if  not (LV = x) .

  -- for turn (x, n, S):
  eq  c-turn (x, n, S) = (which (x, S) = r  and  not (using (s1, S)))
                      or  (which (x, S) = l  and  which (y, S) = n) .
  ceq e-which-turn (LV, x, n, S) = n                 if       LV = x .
  ceq e-which-turn (LV, x, n, S) = which (LV, S)     if  not (LV = x) .

  -- for turn (x, r, S):
  eq  c-turn (x, r, S) = which (x, S) = n
                     and  which (y, S) = r .
  ceq e-which-turn (LV, x, r, S) = r                 if       LV = x .
  ceq e-which-turn (LV, x, r, S) = which (LV, S)     if  not (LV = x) .

  -- for turn (y, l, S):
  eq  c-turn (y, l, S) = which (x, S) = l
                     and  which (y, S) = n .
  ceq e-which-turn (LV, y, l, S) = l                 if       LV = y .
  ceq e-which-turn (LV, y, l, S) = which (LV, S)     if  not (LV = y) .

  -- for turn (y, n, S):
  eq  c-turn (y, n, S) = (which (y, S) = l  and  not (using (s4, S)))
                     or   (which (y, S) = r  and  which (x, S) = n) .
  ceq e-which-turn (LV, y, n, S) = n                 if       LV = y .
  ceq e-which-turn (LV, y, n, S) = which (LV, S)     if  not (LV = y) .

  -- for turn (y, r, S):
  eq  c-turn (y, r, S) = which (x, S) = n
                     and  which (y, S) = n .
  ceq e-which-turn (LV, y, r, S) = r                 if       LV = y .
  ceq e-which-turn (LV, y, r, S) = which (LV, S)     if  not (LV = y) .

  -- generic behavior for turn (LV', LD, S).
  eq  signal (SG, turn (LV', LD, S)) = signal (SG, S) .
  eq  num (TC, turn (LV', LD, S)) = num (TC, S) .
  eq  pos (TR, turn (LV', LD, S)) = pos (TR, S) .
  eq  dir (TR, turn (LV', LD, S)) = dir (TR, S) .
  ceq which (LV, turn (LV', LD, S)) = e-which-turn (LV, LV', LD, S)
      if  c-turn (LV', LD, S) .
  ceq which (LV, turn (LV', LD, S)) = which (LV, S)
      if  not (c-turn (LV', LD, S)) .
  eq  using (SG, turn (LV', LD, S)) = using (SG, S) .

-- ------------------------------------------------------------------------
```

```
-- ------------------------------------------------------------------------

  op  c-allow        :       SgID State -> Bool
  op  e-signal-allow : SgID  SgID State -> Bool
  op  e-using-allow  : SgID  SgID State -> Bool

  -- for allow (s1, S):
  eq  c-allow (s1, S) = which (x, S) = r
                    and  which (y, S) = r
                    and  not (num (t3, S) > 0)
                    and  not (num (t4, S) > 0)
                    and  not (using (s1, S))
                    and  not (using (s2, S)) .
  ceq e-signal-allow (SG, s1, S) = true              if       SG = s1 .
  ceq e-signal-allow (SG, s1, S) = signal (SG, S)    if  not (SG = s1) .
  ceq e-using-allow (SG, s1, S) = true               if       SG = s1 .
  ceq e-using-allow (SG, s1, S) = using (SG, S)      if  not (SG = s1) .

  -- for allow (s2, S):
  eq  c-allow (s2, S) = not (num (t2, S) > 0)
                    and  not (num (t3, S) > 0)
                    and  not (using (s1, S))
                    and  not (using (s2, S)) .
  ceq e-signal-allow (SG, s2, S) = true              if       SG = s2 .
  ceq e-signal-allow (SG, s2, S) = signal (SG, S)    if  not (SG = s2) .
  ceq e-using-allow (SG, s2, S) = true               if       SG = s2 .
  ceq e-using-allow (SG, s2, S) = using (SG, S)      if  not (SG = s2) .

  -- for allow (s3, S):
  eq  c-allow (s3, S) = not (num (t5, S) > 0)
                    and  not (num (t6, S) > 0)
                    and  not (using (s3, S))
                    and  not (using (s4, S)) .
  ceq e-signal-allow (SG, s3, S) = true              if       SG = s3 .
  ceq e-signal-allow (SG, s3, S) = signal (SG, S)    if  not (SG = s3) .
  ceq e-using-allow (SG, s3, S) = true               if       SG = s3 .
  ceq e-using-allow (SG, s3, S) = using (SG, S)      if  not (SG = s3) .

  -- for allow (s4, S):
  eq  c-allow (s4, S) = which (x, S) = l
                    and  which (y, S) = l
                    and  not (num (t4, S) > 0)
                    and  not (num (t5, S) > 0)
                    and  not (using (s3, S))
                    and  not (using (s4, S)) .
  ceq e-signal-allow (SG, s4, S) = true               if       SG = s4 .
  ceq e-signal-allow (SG, s4, S) = signal (SG, S)     if  not (SG = s4) .
  ceq e-using-allow (SG, s4, S) = true                if       SG = s4 .
  ceq e-using-allow (SG, s4, S) = using (SG, S)       if  not (SG = s4) .

  -- generic behavior for allow (SG, S).
  ceq signal (SG, allow (SG', S)) = e-signal-allow (SG, SG', S)
      if  c-allow (SG', S) .
```

```
    ceq signal (SG, allow (SG', S)) = signal (SG, S)
        if not (c-allow (SG', S)) .
    eq  num (TC, allow (SG', S)) = num (TC, S) .
    eq  pos (TR, allow (SG', S)) = pos (TR, S) .
    eq  dir (TR, allow (SG', S)) = dir (TR, S) .
    eq  which (LV, allow (SG', S)) = which (LV, S) .
    ceq using (SG, allow (SG', S)) = e-using-allow (SG, SG', S)
        if  c-allow (SG', S) .
    ceq using (SG, allow (SG', S)) = using (SG, S)
        if  not (c-allow (SG', S)) .

-- ---------------------------------------------------------------
-- ---------------------------------------------------------------

    op  c-forbid          :     SgID State -> Bool
    op  e-signal-forbid : SgID  SgID State -> Bool
    op  e-using-forbid  : SgID  SgID State -> Bool

    -- for forbid (s1, S):
    eq  c-forbid (s1, S) = using (s1, S)
                       and not (num (t3, S) > 0) .
    ceq e-signal-forbid (SG, s1, S) = false
                                          if     SG = s1 .
    ceq e-signal-forbid (SG, s1, S) = signal (SG, S)
                                          if  not (SG = s1) .
    ceq e-using-forbid (SG, s1, S) = using (SG, S)
                                          if  not (SG = s1) .

    -- for forbid (s2, S):
    eq  c-forbid (s2, S) = using (s2, S)
                       and not (num (t3, S) > 0) .
    ceq e-signal-forbid (SG, s2, S) = false
                                          if     SG = s2 .
    ceq e-signal-forbid (SG, s2, S) = signal (SG, S)
                                          if  not (SG = s2) .
    ceq e-using-forbid (SG, s2, S) = false
                                          if     SG = s2 .
    ceq e-using-forbid (SG, s2, S) = using (SG, S)
                                          if  not (SG = s2) .

    -- for forbid (s3, S):
    eq  c-forbid (s3, S) = using (s3, S)
                       and not (num (t5, S) > 0) .
    ceq e-signal-forbid (SG, s3, S) = false
                                          if     SG = s3 .
    ceq e-signal-forbid (SG, s3, S) = signal (SG, S)
                                          if  not (SG = s3) .
    ceq e-using-forbid (SG, s3, S) = false
                                          if     SG = s3 .
    ceq e-using-forbid (SG, s3, S) = using (SG, S)
                                          if  not (SG = s3) .

    -- for forbid (s4, S):
    eq  c-forbid (s4, S) = using (s4, S)
                       and not (num (t5, S) > 0) .
    ceq e-signal-forbid (SG, s4, S) = false
                                          if     SG = s4 .
    ceq e-signal-forbid (SG, s4, S) = signal (SG, S)
                                          if  not (SG = s4) .
    ceq e-using-forbid (SG, s4, S) = false
                                          if     SG = s4 .
    ceq e-using-forbid (SG, s4, S) = using (SG, S)
                                          if  not (SG = s4) .

    -- generic behavior for forbid (SG, S).
    ceq signal (SG, forbid (SG', S)) = e-signal-forbid (SG, SG', S)
        if  c-forbid (SG', S) .
```

```
    ceq signal (SG, forbid (SG', S)) = signal (SG, S)
        if not (c-forbid (SG', S)) .
    eq  num (TC, forbid (SG', S)) = num (TC, S) .
    eq  pos (TR, forbid (SG', S)) = pos (TR, S) .
    eq  dir (TR, forbid (SG', S)) = dir (TR, S) .
    eq  which (LV, forbid (SG', S)) = which (LV, S) .
    ceq using (SG, forbid (SG', S)) = e-using-forbid (SG, SG', S)
        if  c-forbid (SG', S) .
    ceq using (SG, forbid (SG', S)) = using (SG, S)
        if not (c-forbid (SG', S)) .
}
```

C.2 検証

C.2.1 原始隠蔽定数の定義

```
mod CASES { pr (ABE)

  *[ Atom < Case < State ]*

    op  s    : -> Case
    op  _o_  : Case Case -> Case  { coherent assoc comm }
    op comp : Atom Case -> Bool  { coherent }
    op tbl  : Atom -> Case       { coherent }
    op _::_ : Case Case -> Case  { coherent assoc comm idem }
    op isin : Case Case -> Bool  { coherent }

    vars A A' : Atom
    vars C C' C'' : Case

    eq  comp (A, C) = not (isin (C, tbl (A)) == true) .
    eq  isin (A,     A)       = true .
    eq  isin (A o C, A)       = true .
    eq  isin (A,     A :: C') = true .
    eq  isin (A o C, A :: C') = true .

    -- ---------------------------------------------------------------

    -- for any train.
    ops c1 c2 : -> TrID
    eq  (c1 = c2) = false .

    var TC : TcID
    var TD : TrDir
    var LD : LvDir

    -- ---------------------------------------------------------------

    -- for pos (c1, s) : State -> { t1, t1, ... , t7, tr }
    ops  cit1  cit2  cit3  cit4  cit5  cit6  cit7  citr : -> Atom
    ops ~cit1 ~cit2 ~cit3 ~cit4 ~cit5 ~cit6 ~cit7 ~citr : -> Atom

    ceq (pos (c1, ( cit1 o C) = TC) = TC = t1  if  comp ( cit1, C) .
    ceq (pos (c1, (~cit1 o C) = t1) = false    if  comp (~cit1, C) .
    ceq (pos (c1, ( cit1 o C) = TC) = TC = t1  if  comp ( cit1, C) .
    ceq (pos (c1, (~cit1 o C) = t1) = false    if  comp (~cit1, C) .
    ceq (pos (c1, ( cit2 o C) = TC) = TC = t2  if  comp ( cit2, C) .
    ceq (pos (c1, (~cit2 o C) = t2) = false    if  comp (~cit2, C) .
    ceq (pos (c1, ( cit3 o C) = TC) = TC = t3  if  comp ( cit3, C) .
    ceq (pos (c1, (~cit3 o C) = t3) = false    if  comp (~cit3, C) .
    ceq (pos (c1, ( cit4 o C) = TC) = TC = t4  if  comp ( cit4, C) .
    ceq (pos (c1, (~cit4 o C) = t4) = false    if  comp (~cit4, C) .
    ceq (pos (c1, ( cit5 o C) = TC) = TC = t5  if  comp ( cit5, C) .
    ceq (pos (c1, (~cit5 o C) = t5) = false    if  comp (~cit5, C) .
    ceq (pos (c1, ( cit6 o C) = TC) = TC = t6  if  comp ( cit6, C) .
    ceq (pos (c1, (~cit6 o C) = t6) = false    if  comp (~cit6, C) .
    ceq (pos (c1, ( cit7 o C) = TC) = TC = t7  if  comp ( cit7, C) .
    ceq (pos (c1, (~cit7 o C) = t7) = false    if  comp (~cit7, C) .
    ceq (pos (c1, ( citr o C) = TC) = TC = tr  if  comp ( citr, C) .
    ceq (pos (c1, (~citr o C) = tr) = false    if  comp (~citr, C) .
```

```
eq  tbl ( c1t1) = ~c1t1 :: c1t1 :: c1t2
                :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t1) = c1t1 .
eq  tbl ( c1t1) = ~c1t1 :: c1t2
                :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t1) = c1t1 .
eq  tbl ( c1t2) = c1t1 :: ~c1t2
                :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t2) = c1t2 .
eq  tbl ( c1t3) = c1t1 :: c1t2
                :: ~c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t3) = c1t3 .
eq  tbl ( c1t4) = c1t1 :: c1t2
                :: c1t3 :: ~c1t4 :: c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t4) = c1t4 .
eq  tbl ( c1t5) = c1t1 :: c1t2
                :: c1t3 :: c1t4 :: ~c1t5 :: c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t5) = c1t5 .
eq  tbl ( c1t6) = c1t1 :: c1t2
                :: c1t3 :: c1t4 :: c1t5 :: ~c1t6 :: c1t7 :: c1tr .
eq  tbl (~c1t6) = c1t6 .
eq  tbl ( c1t7) = c1t1 :: c1t2
                :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: ~c1t7 :: c1tr .
eq  tbl (~c1t7) = c1t7 .
eq  tbl ( c1tr) = c1t1 :: c1t2
                :: c1t3 :: c1t4 :: c1t5 :: c1t6 :: c1t7 :: ~c1tr .
eq  tbl (~c1tr) = c1tr .

-- for pos (c2, s) : State -> { t1, t1, ... , t7, tr }
ops  c2t1  c2t2  c2t3  c2t4  c2t5  c2t6  c2t7  c2tr : -> Atom
ops ~c2t1 ~c2t2 ~c2t3 ~c2t4 ~c2t5 ~c2t6 ~c2t7 ~c2tr : -> Atom

ceq (pos (c2, ( c2l o c)) = TC) = TC = tl  if  comp ( c2l, c) .
ceq (pos (c2, (~c2l o c)) = tl) = false    if  comp (~c2l, c) .
ceq (pos (c2, ( c2t o c)) = TC) = TC = t1  if  comp ( c2t, c) .
ceq (pos (c2, (~c2t o c)) = t1) = false    if  comp (~c2t, c) .
ceq (pos (c2, ( c2r o c)) = TC) = TC = t2  if  comp ( c2r, c) .
ceq (pos (c2, (~c2r o c)) = t2) = false    if  comp (~c2r, c) .
ceq (pos (c2, ( c2t3 o c)) = TC) = TC = t3  if  comp ( c2t3, c) .
ceq (pos (c2, (~c2t3 o c)) = t3) = false    if  comp (~c2t3, c) .
ceq (pos (c2, ( c2t4 o c)) = TC) = TC = t4  if  comp ( c2t4, c) .
ceq (pos (c2, (~c2t4 o c)) = t4) = false    if  comp (~c2t4, c) .
ceq (pos (c2, ( c2t5 o c)) = TC) = TC = t5  if  comp ( c2t5, c) .
ceq (pos (c2, (~c2t5 o c)) = t5) = false    if  comp (~c2t5, c) .
ceq (pos (c2, ( c2t6 o c)) = TC) = TC = t6  if  comp ( c2t6, c) .
ceq (pos (c2, (~c2t6 o c)) = t6) = false    if  comp (~c2t6, c) .
ceq (pos (c2, ( c2t7 o c)) = TC) = TC = t7  if  comp ( c2t7, c) .
ceq (pos (c2, (~c2t7 o c)) = t7) = false    if  comp (~c2t7, c) .
ceq (pos (c2, ( c2tr o c)) = TC) = TC = tr  if  comp ( c2tr, c) .
ceq (pos (c2, (~c2tr o c)) = tr) = false    if  comp (~c2tr, c) .
```

```
eq  tbl ( c2t1) = ~c2t1 :: c2t1 :: c2t2
                :: c2t3 :: c2t4 :: c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t1) = c2t1 .
eq  tbl ( c2t1) = ~c2t1 :: c2t2
                :: c2t3 :: c2t4 :: c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t1) = c2t1 .
eq  tbl ( c2t2) = c2t1 :: ~c2t2
                :: c2t3 :: c2t4 :: c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t2) = c2t2 .
eq  tbl ( c2t3) = c2t1 :: c2t2
                :: ~c2t3 :: c2t4 :: c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t3) = c2t3 .
eq  tbl ( c2t4) = c2t1 :: c2t2
                :: c2t3 :: ~c2t4 :: c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t4) = c2t4 .
eq  tbl ( c2t5) = c2t1 :: c2t2
                :: c2t3 :: c2t4 :: ~c2t5 :: c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t5) = c2t5 .
eq  tbl ( c2t6) = c2t1 :: c2t2
                :: c2t3 :: c2t4 :: c2t5 :: ~c2t6 :: c2t7 :: c2tr .
eq  tbl (~c2t6) = c2t6 .
eq  tbl ( c2t7) = c2t1 :: c2t2
                :: c2t3 :: c2t4 :: c2t5 :: c2t6 :: ~c2t7 :: c2tr .
eq  tbl (~c2t7) = c2t7 .
eq  tbl ( c2tr) = c2t1 :: c2t2
                :: c2t3 :: c2t4 :: c2t5 :: c2t6 :: c2t7 :: ~c2tr .
eq  tbl (~c2tr) = c2tr .

-- for dir (c1, s) : State -> { r, l }
ops  c1r  c1l : -> Atom
ops ~c1r ~c1l : -> Atom

ceq (dir (c1, ( c1r o c)) = TD) = TD = r  if  comp ( c1r, c) .
ceq (dir (c1, (~c1r o c)) = r)  = false   if  comp (~c1r, c) .
ceq (dir (c1, ( c1l o c)) = TD) = TD = l  if  comp ( c1l, c) .
ceq (dir (c1, (~c1l o c)) = l)  = false   if  comp (~c1l, c) .

eq  tbl ( c1r) = ~c1r :: c1l .
eq  tbl (~c1r) =  c1r .
eq  tbl ( c1l) =  c1r :: ~c1l .
eq  tbl (~c1l) =  c1l .

-- for dir (c2, s) : State -> { r, l }
ops  c2r  c2l : -> Atom
ops ~c2r ~c2l : -> Atom

ceq (dir (c2, ( c2r o c)) = TD) = TD = r  if  comp ( c2r, c) .
ceq (dir (c2, (~c2r o c)) = r)  = false   if  comp (~c2r, c) .
```

```
ceq (dir (c2, ( c2l o C)) = TD) = TD = l   if  comp ( c2l, C) .
ceq (dir (c2, (~c2l o C)) = l)  = false    if  comp (~c2l, C) .

eq  tbl ( c2r) = ~c2r ::  c2l .
eq  tbl (~c2r) =  c2r .
eq  tbl ( c2l) =  c2r :: ~c2l .
eq  tbl (~c2l) =  c2l .

-- ---------------------------------------------------------------------

-- for num (t1, s) : State -> Nat
ops  t1nz ~t1nz : -> Atom

ceq (num (t1, ( t1nz o C)) > 0) = true    if  comp ( t1nz, C) .
ceq (num (t1, (~t1nz o C)) > 0) = false   if  comp (~t1nz, C) .

eq  tbl ( t1nz) = ~t1nz .
eq  tbl (~t1nz) =  t1nz .

-- ---------------------------------------------------------------------

-- for num (t2, s) : State -> Nat
ops  t2nz ~t2nz : -> Atom

ceq (num (t2, ( t2nz o C)) > 0) = true    if  comp ( t2nz, C) .
ceq (num (t2, (~t2nz o C)) > 0) = false   if  comp (~t2nz, C) .

eq  tbl ( t2nz) = ~t2nz .
eq  tbl (~t2nz) =  t2nz .

-- ---------------------------------------------------------------------

ops  t3nz ~t3nz : -> Atom

ceq (num (t3, ( t3nz o C)) > 0) = true    if  comp ( t3nz, C) .
ceq (num (t3, (~t3nz o C)) > 0) = false   if  comp (~t3nz, C) .

eq  tbl ( t3nz) = ~t3nz .
eq  tbl (~t3nz) =  t3nz .

-- ---------------------------------------------------------------------

ops  t4nz ~t4nz : -> Atom

ceq (num (t4, ( t4nz o C)) > 0) = true    if  comp ( t4nz, C) .
ceq (num (t4, (~t4nz o C)) > 0) = false   if  comp (~t4nz, C) .

eq  tbl ( t4nz) = ~t4nz .
eq  tbl (~t4nz) =  t4nz .

-- ---------------------------------------------------------------------

ops  t5nz ~t5nz : -> Atom
```

```
ceq (num (t5, ( t5nz o C)) > 0) = true    if  comp ( t5nz, C) .
ceq (num (t5, (~t5nz o C)) > 0) = false   if  comp (~t5nz, C) .

eq  tbl ( t5nz) = ~t5nz .
eq  tbl (~t5nz) =  t5nz .

-- ---------------------------------------------------------------------

ops  t6nz ~t6nz : -> Atom

ceq (num (t6, ( t6nz o C)) > 0) = true    if  comp ( t6nz, C) .
ceq (num (t6, (~t6nz o C)) > 0) = false   if  comp (~t6nz, C) .

eq  tbl ( t6nz) = ~t6nz .
eq  tbl (~t6nz) =  t6nz .

-- ---------------------------------------------------------------------

ops  t7nz ~t7nz : -> Atom

ceq (num (t7, ( t7nz o C)) > 0) = true    if  comp ( t7nz, C) .
ceq (num (t7, (~t7nz o C)) > 0) = false   if  comp (~t7nz, C) .

eq  tbl ( t7nz) = ~t7nz .
eq  tbl (~t7nz) =  t7nz .

-- ---------------------------------------------------------------------

-- for sig (s1, s) : State -> Bool
-- for sig (s2, s) : State -> Bool
-- for sig (s3, s) : State -> Bool
-- for sig (s4, s) : State -> Bool
ops  s1g  s1r  s2g  s2r  s3g  s3r  s4g  s4r : -> Atom
ops ~s1g ~s1r ~s2g ~s2r ~s3g ~s3r ~s4g ~s4r : -> Atom

eq  ~s1g = s1r .
eq  ~s1r = s1g .
eq  ~s2g = s2r .
eq  ~s2r = s2g .
eq  ~s3g = s3r .
eq  ~s3r = s3g .
eq  ~s4g = s4r .
eq  ~s4r = s4g .

ceq (signal (s1, s1g o C)) = true   if  comp ( s1g, C) .
ceq (signal (s1, s1r o C)) = false  if  comp ( s1r, C) .
ceq (signal (s2, s2g o C)) = true   if  comp ( s2g, C) .
ceq (signal (s2, s2r o C)) = false  if  comp ( s2r, C) .
ceq (signal (s3, s3g o C)) = true   if  comp ( s3g, C) .
ceq (signal (s3, s3r o C)) = false  if  comp ( s3r, C) .
ceq (signal (s4, s4g o C)) = true   if  comp ( s4g, C) .
ceq (signal (s4, s4r o C)) = false  if  comp ( s4r, C) .
```

```
eq  tbl ( s1g) =  s1r .
eq  tbl ( s1r) =  s1g .
eq  tbl ( s2g) =  s2r .
eq  tbl ( s2r) =  s2g .
eq  tbl ( s3g) =  s3r .
eq  tbl ( s3r) =  s3g .
eq  tbl ( s4g) =  s4r .
eq  tbl ( s4r) =  s4g .

-- ---------------------------------------------------------------------

-- for which (x, s) : State -> { l, n, r }
ops  txl  txn  txr : -> Atom
ops ~txl ~txn ~txr : -> Atom

ceq (which (x, ( txl o C)) = LD) = LD = l    if  comp ( txl, C) .
ceq (which (x, (~txl o C)) = l)  = false     if  comp (~txl, C) .
ceq (which (x, ( txn o C)) = LD) = LD = n    if  comp ( txn, C) .
ceq (which (x, (~txn o C)) = n)  = false     if  comp (~txn, C) .
ceq (which (x, ( txr o C)) = LD) = LD = r    if  comp ( txr, C) .
ceq (which (x, (~txr o C)) = r)  = false     if  comp (~txr, C) .

eq  tbl ( txl) =  txn :: txr .
eq  tbl (~txl) =  txl .
eq  tbl ( txn) =  txl :: txr .
eq  tbl (~txn) =  txn .
eq  tbl ( txr) =  txl :: txn .
eq  tbl (~txr) =  txr .

-- ---------------------------------------------------------------------

-- for which (x, s) : State -> { l, n, r }
ops  tyl  tyn  tyr : -> Atom
ops ~tyl ~tyn ~tyr : -> Atom

ceq (which (y, ( tyl o C)) = LD) = LD = l    if  comp ( tyl, C) .
ceq (which (y, (~tyl o C)) = l)  = false     if  comp (~tyl, C) .
ceq (which (y, ( tyn o C)) = LD) = LD = n    if  comp ( tyn, C) .
ceq (which (y, (~tyn o C)) = n)  = false     if  comp (~tyn, C) .
ceq (which (y, ( tyr o C)) = LD) = LD = r    if  comp ( tyr, C) .
ceq (which (y, (~tyr o C)) = r)  = false     if  comp (~tyr, C) .

eq  tbl ( tyl) =  tyn :: tyr .
eq  tbl (~tyl) =  tyl .
eq  tbl ( tyn) =  tyl :: tyr .
eq  tbl (~tyn) =  tyn .
eq  tbl ( tyr) =  tyl :: tyn .
eq  tbl (~tyr) =  tyr .

-- ---------------------------------------------------------------------

-- for using (s1, s) : State -> Bool
```

```
  -- for using (s2, s) : State -> Bool
  -- for using (s3, s) : State -> Bool
  -- for using (s4, s) : State -> Bool
  ops  us1  us2  us3  us4 : -> Atom
  ops ~us1 ~us2 ~us3 ~us4 : -> Atom

  ceq (using (s1,  us1 o C)) = true   if  comp ( us1, C) .
  ceq (using (s1, ~us1 o C)) = false  if  comp (~us1, C) .
  ceq (using (s2,  us2 o C)) = true   if  comp ( us2, C) .
  ceq (using (s2, ~us2 o C)) = false  if  comp (~us2, C) .
  ceq (using (s3,  us3 o C)) = true   if  comp ( us3, C) .
  ceq (using (s3, ~us3 o C)) = false  if  comp (~us3, C) .
  ceq (using (s4,  us4 o C)) = true   if  comp ( us4, C) .
  ceq (using (s4, ~us4 o C)) = false  if  comp (~us4, C) .

  eq  tbl ( us1) = ~us1 .
  eq  tbl (~us1) =  us1 .
  eq  tbl ( us2) = ~us2 .
  eq  tbl (~us2) =  us2 .
  eq  tbl ( us3) = ~us3 .
  eq  tbl (~us3) =  us3 .
  eq  tbl ( us4) = ~us4 .
  eq  tbl (~us4) =  us4 .
}
```

C.2.2 帰納法の検証の定義

```
-- ==========================================================================
--  proof.mod  -- ver.
--
--      a single-track railroad signaling system
--                      with automatic blocking system (type-B).
-- --------------------------------------------------------------------------
-- Original Copyright (c) 2000-2002 Takahiro Seino, all rights reserved.
-- ==========================================================================

mod PROOF { ex (CASES) pr (STRING)
  op  p : State -> Bool  -- the predicate to be proven.
  ops INI IND : -> Bool  -- grand terms of the proof.

  op  _|_ : Bool Bool -> Bool { assoc comm idem }

  op  comps : Case Case -> Bool
  eq  comps (A:Atom, C':Case)
            = comp (A, C') .
  eq  comps (A:Atom o A':Atom, C':Case)
            = comp (A, C') and-also comp (A', C') .
  eq  comps (A:Atom o A':Atom o C:Case, C':Case)
            = comp (A, C') and-also comps (A' o C, C') .

-- ==========================================================================
--  tricks for convinience and efficency.
-- ==========================================================================

  op _ => _ : Bool Bool -> Bool { strat: (1 0 2) prec: 61 }
  eq B1:Bool => B2:Bool = (not B1) or-else B2 .

  -- allows to add a simple comment in equation.
  op _ -- _  : Bool String -> Bool { strat: (0) prec: 20 }
  eq B:Bool --  S:String = B .

  -- allows to ingore the boolean expresion with comment.
  op _ --> _ : Bool String -> Bool { strat: (0) prec: 20 }
  eq B:Bool --> S:String = true .

-- ==========================================================================
--  for all actions:
-- --------------------------------------------------------------------------

  op  MOVE  : TrID TcID TrDir Case Case -> Bool
  ceq MOVE (TR:TrID, TC:TcID, TD:TrDir, C:Case, C':Case) = true
    if  comps (C, C') => (p (C o C') => p (move (TR, TC, TD, (C o C')))) .

  op  TURN  : LvID LvDir Case Case -> Bool
  ceq TURN (LV:LvID, LD:LvDir, C:Case, C':Case) = true
    if  comps (C, C') => (p (C o C') => p (turn (LV, LD, (C o C')))) .

  op  ALLOW  : SgID Case Case -> Bool
  ceq ALLOW (SG:SgID, C:Case, C':Case) = true
    if  comps (C, C') => (p (C o C') => p (allow (SG, (C o C')))) .

  op  FORBID : SgID Case Case -> Bool
  ceq FORBID (SG:SgID, C:Case, C':Case) = true
    if  comps (C, C') => (p (C o C') => p (forbid (SG, (C o C')))) .

  op  FORALL-ACTION : Case -> Bool { memo }

-- ==========================================================================
--  for analysis:
-- --------------------------------------------------------------------

  op  traverse : Pairlist Case -> Pairlist
  op  try : Atom Case -> Atom
  eq  traverse (empty, C:Case) = empty .
  eq  traverse (<< A1:Atom, A2:Atom >> :: L:Pairlist, C:Case)
        = << try (A1, C), try (A2, C) >> :: traverse (L, C) .

  bop act : State -> State

  ops tt ff : -> Atom
  eq  try (A:Atom, C:Case)
        = if  (comp (A, C) => (p (A o C) => p (act (A o C)))) == true
              then  tt
              else  if  (comp (A, C) => (p (A o C) => p (act (A o C)))) == false
                        then  ff
                        else  A
                    fi
        fi .
}
```

```
-- In arbitrary train c1:
mod PROOF1 { ex (PROOF)
  eq  FORALL-ACTION (HYP:Case)
        = MOVE (c1, t1, r, c1t1 o c1r o ~t1nz, HYP)
        | MOVE (c1, t1, r, c1t1 o c1r o s1g,   HYP)
        | MOVE (c1, t2, r, c1t2 o c1r,         HYP)
        | MOVE (c1, t3, r, c1t3 o c1r,         HYP)
        | MOVE (c1, t4, r, c1t4 o c1r o s3g,   HYP)
        | MOVE (c1, t5, r, c1t5 o c1r,         HYP)
        | MOVE (c1, t6, r, c1t6 o c1r,         HYP)
        | MOVE (c1, t7, r, c1t7 o c1r,         HYP)
        | MOVE (c1, tr, r, c1tr o c1r,         HYP)

        | MOVE (c1, t1, l, c1t1 o c1l,         HYP)
        | MOVE (c1, t1, l, c1t1 o c1l,         HYP)
        | MOVE (c1, t2, l, c1t2 o c1l,         HYP)
        | MOVE (c1, t3, l, c1t3 o c1l,         HYP)
        | MOVE (c1, t4, l, c1t4 o c1l o s2g,   HYP)
        | MOVE (c1, t5, l, c1t5 o c1l,         HYP)
        | MOVE (c1, t6, l, c1t6 o c1l,         HYP)
        | MOVE (c1, t7, l, c1t7 o c1l o s4g,   HYP)
        | MOVE (c1, tr, l, c1tr o c1l o ~t7nz, HYP)
```

```
       | TURN (x, l, txn o  tyn, HYP)
       | TURN (x, n, txr o ~us1, HYP) | TURN (x, n, txl o tyn, HYP)
       | TURN (x, r, txn o  tyr, HYP)

       | TURN (y, l, txn o  tyr, HYP)
       | TURN (y, n, tyl o ~us4, HYP) | TURN (y, n, tyr o txn, HYP)
       | TURN (y, r, txn o  tyn, HYP)

       | ALLOW (s1, txr o tyr o ~t3nz o ~t4nz o ~us1 o ~us2, HYP)
       | ALLOW (s2,             ~t2nz o ~t3nz o ~us1 o ~us2, HYP)
       | ALLOW (s3,             ~t5nz o ~t6nz o ~us3 o ~us4, HYP)
       | ALLOW (s4, txl o tyl o ~t4nz o ~t5nz o ~us3 o ~us4, HYP)

       | FORBID (s1, us1 o ~t3nz, HYP)
       | FORBID (s2, us2 o ~t3nz, HYP)
       | FORBID (s3, us3 o ~t5nz, HYP)
       | FORBID (s4, us4 o ~t5nz, HYP) .
}

-- In arbitrary train c1 and c2:
mod PROOF2 { ex (PROOF)
  eq  FORALL-ACTION (HYP:Case)
       = MOVE (c1, tl, r, c1tl o c1r o ~t1nz, HYP)
       | MOVE (c1, t1, r, c1t1 o c1r o s1g,   HYP)
       | MOVE (c1, t2, r, c1t2 o c1r,         HYP)
       | MOVE (c1, t3, r, c1t3 o c1r,         HYP)
       | MOVE (c1, t4, r, c1t4 o c1r o s3g,   HYP)
       | MOVE (c1, t5, r, c1t5 o c1r,         HYP)
       | MOVE (c1, t6, r, c1t6 o c1r,         HYP)
       | MOVE (c1, t7, r, c1t7 o c1r,         HYP)
       | MOVE (c1, tr, r, c1tr o c1r,         HYP)

       | MOVE (c1, tl, l, c1tl o c1l,         HYP)
       | MOVE (c1, t1, l, c1t1 o c1l,         HYP)
       | MOVE (c1, t2, l, c1t2 o c1l,         HYP)
       | MOVE (c1, t3, l, c1t3 o c1l,         HYP)
       | MOVE (c1, t4, l, c1t4 o c1l o s2g,   HYP)
       | MOVE (c1, t5, l, c1t5 o c1l,         HYP)
       | MOVE (c1, t6, l, c1t6 o c1l,         HYP)
       | MOVE (c1, t7, l, c1t7 o c1l o s4g,   HYP)
       | MOVE (c1, tr, l, c1tr o c1l o ~t7nz, HYP)

       | MOVE (c2, tl, r, c2tl o c2r o ~t1nz, HYP)
       | MOVE (c2, t1, r, c2t1 o c2r o s1g,   HYP)
       | MOVE (c2, t2, r, c2t2 o c2r,         HYP)
       | MOVE (c2, t3, r, c2t3 o c2r,         HYP)
       | MOVE (c2, t4, r, c2t4 o c2r o s3g,   HYP)
       | MOVE (c2, t5, r, c2t5 o c2r,         HYP)
       | MOVE (c2, t6, r, c2t6 o c2r,         HYP)
       | MOVE (c2, t7, r, c2t7 o c2r,         HYP)
       | MOVE (c2, tr, r, c2tr o c2r,         HYP)

       | MOVE (c2, tl, l, c2tl o c2l,         HYP)
```

```
       | MOVE (c2, t1, l, c2t1 o c2l,         HYP)
       | MOVE (c2, t2, l, c2t2 o c2l,         HYP)
       | MOVE (c2, t3, l, c2t3 o c2l,         HYP)
       | MOVE (c2, t4, l, c2t4 o c2l o s2g,   HYP)
       | MOVE (c2, t5, l, c2t5 o c2l,         HYP)
       | MOVE (c2, t6, l, c2t6 o c2l,         HYP)
       | MOVE (c2, t7, l, c2t7 o c2l o s4g,   HYP)
       | MOVE (c2, tr, l, c2tr o c2l o ~t7nz, HYP)

       | TURN (x, l, txn o  tyn, HYP)
       | TURN (x, n, txr o ~us1, HYP) | TURN (x, n, txl o tyn, HYP)
       | TURN (x, r, txn o  tyr, HYP)

       | TURN (y, l, txn o  tyr, HYP)
       | TURN (y, n, tyl o ~us4, HYP) | TURN (y, n, tyr o txn, HYP)
       | TURN (y, r, txn o  tyn, HYP)

       | ALLOW (s1, txr o tyr o ~t3nz o ~t4nz o ~us1 o ~us2, HYP)
       | ALLOW (s2,             ~t2nz o ~t3nz o ~us1 o ~us2, HYP)
       | ALLOW (s3,             ~t5nz o ~t6nz o ~us3 o ~us4, HYP)
       | ALLOW (s4, txl o tyl o ~t4nz o ~t5nz o ~us3 o ~us4, HYP)

       | FORBID (s1, us1 o ~t3nz, HYP)
       | FORBID (s2, us2 o ~t3nz, HYP)
       | FORBID (s3, us3 o ~t5nz, HYP)
       | FORBID (s4, us4 o ~t5nz, HYP) .
}
```

C.2.3 補題 6.A.1 から 6.A.15

```
mod LEMMA6-A-1 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t1)
        or-else  (num (t1, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t1 .
  eq  hyp2 =  t1nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
      | FORALL-ACTION (hyp2) .
}
mod LEMMA6-A-2 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t2)
        or-else  (num (t2, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t2 .
  eq  hyp2 =  t2nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
      | FORALL-ACTION (hyp2) .
}
mod LEMMA6-A-3 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t3)
        or-else  (num (t3, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 =  t3nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
      | FORALL-ACTION (hyp2) .
}
mod LEMMA6-A-4 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t4)
        or-else  (num (t4, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t4 .
  eq  hyp2 =  t4nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
      | FORALL-ACTION (hyp2) .
}
mod LEMMA6-A-5 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t5)
        or-else  (num (t5, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 =  t5nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
      | FORALL-ACTION (hyp2) .
}
mod LEMMA6-A-6 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t6)
        or-else  (num (t6, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
      | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t6 .
```

```
  eq  hyp2 =  t6nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-7 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t7)
           or-else  (num (t7, S) > 0) .

  -- base step.
  eq  INI = p (c1r o c1tl)
          | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t7 .
  eq  hyp2 =  t7nz .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-8 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s1, S))
           or-else  (using (s1, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s1g .
  eq  hyp2 =  us1 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-9 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s2, S))
           or-else  (using (s2, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
```

```
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s2g .
  eq  hyp2 =  us2 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-10 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s3, S))
           or-else  (using (s3, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s3g .
  eq  hyp2 =  us3 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-11 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s4, S))
           or-else  (using (s4, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s4g .
  eq  hyp2 =  us4 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}

mod LEMMA6-A-12 {
  ex (PROOF1)
  eq p (S:State) = (not signal (s1, S))
           or-else (not signal (s2, S)) .

  -- base case.
  eq  INI = p (init) .
```

```
  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s1g .
  eq  hyp2 = ~s2g .

  -- induction step.

  eq  IND = FORALL-ACTION (hyp1 o ~s2g)
          | FORALL-ACTION (hyp1 o  s2g o us2)  -- "uses the Lemma6-A-9."
          | FORALL-ACTION (hyp2 o ~s1g)
          | FORALL-ACTION (hyp2 o  s1g o us1)  -- "uses the Lemma6-A-8." .
}

mod LEMMA6-A-13 {
  ex (PROOF1)
  eq p (S:State) = (not signal (s3, S))
           or-else (not signal (s4, S)) .

  -- base case.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s3g .
  eq  hyp2 = ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o ~s4g)
          | FORALL-ACTION (hyp1 o  s4g o us4)  -- "uses the Lemma6-A-11."
          | FORALL-ACTION (hyp2 o ~s3g)
          | FORALL-ACTION (hyp2 o  s3g o us3)  -- "uses the Lemma6-A-10." .
}

mod LEMMA6-A-14 {
  ex (PROOF1)
  eq p (S:State) = (not pos (c1, S) = t3)
           or-else (not signal (s1, S)  and  not signal (s2, S)) .

  -- base case.
  eq  INI = p (c1tl o c1r o s1r o s2r)
          | p (c1tr o c1l o s1r o s2r) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~s1g o ~s2g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s1g  o s2r)  -- "uses the Lemma6-A-12."
          | FORALL-ACTION (hyp1 o ~s1g)
          | FORALL-ACTION (hyp2 o  c1t3 o t3nz)  -- "uses the Lemma6-A-3."
          | FORALL-ACTION (hyp2 o ~c1t3) .
}
```

```
mod LEMMA6-A-15 {
  ex (PROOF1)
  eq p (S:State) = (not pos (c1, S) = t5)
           or-else (not signal (s3, S)  and  not signal (s4, S)) .

  -- base case.
  eq  INI = p (c1tl o c1r o s3r o s4r)
          | p (c1tr o c1l o s3r o s4r) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 = ~s3g o ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s4g  o s3r)  -- "uses the Lemma6-A-13."
          | FORALL-ACTION (hyp1 o ~s4g)
          | FORALL-ACTION (hyp2 o  c1t5 o t5nz)  -- "uses the Lemma6-A-5."
          | FORALL-ACTION (hyp2 o ~c1t5) .
}
```

C.2.4　補題 6.B.1 から 6.B.4

```
mod LEMMA6-B-1 {
  ex (PROOF1)
  eq p (S:State) = (not pos (c1, S) = t2)
       or-else (not signal (s2, S)) .

  -- base step.
  eq  INI = p (c1t1 o s2r)
      | p (c1tr o s2r) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t2 .
  eq  hyp2 = ~s2g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t3 o s2r)  -- "uses the Lemma6-A-14."
      | FORALL-ACTION (hyp1 o ~c1t3)
      | FORALL-ACTION (hyp2 o  c1t2 o t2nz)  -- "uses the Lemma6-A-2."
      | FORALL-ACTION (hyp2 o ~c1t2) .
}

mod LEMMA6-B-2 {
  ex (PROOF1)
  eq p (S:State) = (not pos (c1, S) = t6)
       or-else (not signal (s3, S)) .

  -- base step.
  eq  INI = p (c1t1 o s3r)
      | p (c1tr o s3r) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t6 .
  eq  hyp2 = ~s3g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t5 o s3r)  -- "uses the Lemma6-A-15."
      | FORALL-ACTION (hyp1 o ~c1t5)
      | FORALL-ACTION (hyp2 o  c1t6 o t6nz)  -- "uses the Lemma6-A-6."
      | FORALL-ACTION (hyp2 o ~c1t6) .
}

mod LEMMA6-B-3 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t2)
       or-else (not pos (c2, S) = t3)
       or-else (not dir (c2, S) = l) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
      | p (c1tr o c2tr)
      | p (c1tl o c2tl)
      | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t2 .
  eq  hyp2 = ~c2t3 .
  eq  hyp3 = ~c2l .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t3 o ~c2t3)  -- "uses the Claim3."
      | FORALL-ACTION (hyp1 o ~c1t3)
      | FORALL-ACTION (hyp2 o  c1t2 o s2r)  -- "uses the Lemma6-B-1."
      | FORALL-ACTION (hyp2 o ~c1t2)
      | FORALL-ACTION (hyp3) .
}

mod LEMMA6-B-4 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t6)
       or-else (not pos (c2, S) = t5)
       or-else (not dir (c2, S) = r) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
      | p (c1tr o c2tr)
      | p (c1tl o c2tl)
      | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t6 .
  eq  hyp2 = ~c2t5 .
  eq  hyp3 = ~c2r .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t5 o ~c2t5)  -- "uses the Claim5."
      | FORALL-ACTION (hyp1 o ~c1t5)
      | FORALL-ACTION (hyp2 o  c1t6 o s3r)  -- "uses the Lemma6-B-2."
      | FORALL-ACTION (hyp2 o ~c1t6)
      | FORALL-ACTION (hyp3) .
}
```

C.2.5 補題 6.C.1 から 6.C.17

補題 6.C.1

```
mod LEMMA6-C-1 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s1, S))
        or-else  (which (x, S) = r  and  which (y, S) = r) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~sig .
  eq  hyp2 =  txr o tyr .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2 o  sig o  us1)  -- "uses the Lemma6-A-8."
          | FORALL-ACTION (hyp2 o  ~sig) .
}
```

補題 6.C.2

```
mod LEMMA6-C-2 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s4, S))
        or-else  (which (x, S) = l  and  which (y, S) = l) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~s4g .
  eq  hyp2 =  txl o tyl .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2 o  s4g o  us4)  -- "uses the Lemma6-A-11."
          | FORALL-ACTION (hyp2 o  ~s4g) .
}
```

補題 6.C.3

```
mod LEMMA6-C-3 {
  ex (PROOF1)
  eq  p (S:State) = (not signal (s1, S))
        or-else  (not signal (s4, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~sig .
  eq  hyp2 = ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s4g o txl o tyl)  -- "uses the Lemma6-C-2."
          | FORALL-ACTION (hyp1 o  ~s4g)
          | FORALL-ACTION (hyp2 o  sig o txr o tyr)  -- "uses the Lemma6-C-1."
          | FORALL-ACTION (hyp2 o  ~sig) .
}
```

補題 6.C.4

```
mod LEMMA6-C-4 {
  ex (PROOF1)
  eq  p (S:State) = (which (x, S) = r)
        or-else  (not signal (s1, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = txr .
  eq  hyp2 = ~sig .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  sig o  us1)  -- "uses the Lemma6-A-8."
          | FORALL-ACTION (hyp1 o  ~sig)
          | FORALL-ACTION (hyp2) .
}
```

補題 6.C.5

```
mod LEMMA6-C-5 {
  ex (PROOF1)
  eq  p (S:State) = (which (y, S) = l)
        or-else  (not signal (s4, S)) .

  -- base step.
  eq  INI = p (init) .
```

```
  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = tyl .
  eq  hyp2 = ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s4g o  us4)  -- "uses the Lemma6-A-11."
          | FORALL-ACTION (hyp1 o ~s4g)
          | FORALL-ACTION (hyp2) .
}
```

補題 **6.C.6**

```
mod LEMMA6-C-6 {
  ex (PROOF1)
  eq  p (S:State) = (using (s1, S))
           or-else  (not signal (s1, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 =  us1 .
  eq  hyp2 = ~s1g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}
```

補題 **6.C.7**

```
mod LEMMA6-C-7 {
  ex (PROOF1)
  eq  p (S:State) = (using (s4, S))
           or-else  (not signal (s4, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 =  us4 .
  eq  hyp2 = ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1)
          | FORALL-ACTION (hyp2) .
}
```

補題 **6.C.8**

```
mod LEMMA6-C-8 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t3)
           or-else  (not dir (c1, S) = r)
           or-else  (using (s1, S)) .

  -- base step.
  eq  INI = p (c1r o c1tl)
          | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~c1r .
  eq  hyp3 = us1 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  us1)
          | FORALL-ACTION (hyp1 o ~us1 o s1r)  -- "uses the Lemma6-C-6."
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t3 o t3nz)
          | FORALL-ACTION (hyp3 o ~c1t3) .
}
```

補題 **6.C.9**

```
mod LEMMA6-C-9 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t5)
           or-else  (not dir (c1, S) = l)
           or-else  (using (s4, S)) .

  -- base step.
  eq  INI = p (c1r o c1tl)
          | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 = ~c1l .
  eq  hyp3 = us4 .

  -- induction step.
```

```
  eq  IND = FORALL-ACTION (hyp1 o  us4)
          | FORALL-ACTION (hyp1 o ~us4 o s4r)  -- "uses the Lemma6-C-7."
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t5 o t5nz)
          | FORALL-ACTION (hyp3 o ~c1t5) .
}
```

補題 **6.C.10**

```
mod LEMMA6-C-10 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t3)
           or-else  (not dir (c1, S) = r)
           or-else  (which (x, S) = r) .

  -- base step.
  eq  INI = p (c1r o c1tl)
          | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~c1r .
  eq  hyp3 = txr .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  txr)
          | FORALL-ACTION (hyp1 o ~txr o  s1r)
                                              -- "uses the Lemma6-C-4."
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t3 o  c1r o us1)
                                              -- "uses the Lemma6-C-8."
          | FORALL-ACTION (hyp3 o  c1t3 o ~c1r)
          | FORALL-ACTION (hyp3 o ~c1t3) .
}
```

補題 **6.C.11**

```
mod LEMMA6-C-11 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t5)
           or-else  (not dir (c1, S) = l)
           or-else  (which (y, S) = l) .

  -- base step.
  eq  INI = p (c1r o c1tl)
          | p (c1l o c1tr) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 = ~c1l .
  eq  hyp3 = tyl .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  tyl)
          | FORALL-ACTION (hyp1 o ~tyl o  s4r)
                                              -- "uses the Lemma6-C-5."
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t5 o  c1l o us4)
                                              -- "uses the Lemma6-C-9."
          | FORALL-ACTION (hyp3 o  c1t5 o ~c1l)
          | FORALL-ACTION (hyp3 o ~c1t5) .
}
```

補題 **6.C.12**

```
mod LEMMA6-C-12 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t3)
           or-else  (not dir (c1, S) = r)
           or-else  (not signal (s4, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~c1r .
  eq  hyp3 = ~s4g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s4g o  s1r)
                                              -- "uses the Lemma6-C-3."
          | FORALL-ACTION (hyp1 o ~s4g)
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t3 o  c1r o  txr)
                                              -- "uses the Lemma6-C-10."
          | FORALL-ACTION (hyp3 o  c1t3 o ~c1r)
          | FORALL-ACTION (hyp3 o ~c1t3) .
}
```

補題 **6.C.13**

```
mod LEMMA6-C-13 {
```

```
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t5)
          or-else  (not dir (c1, S) = l)
          or-else  (not signal (s1, S)) .

  -- base step.
  eq  INI = p (init) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 = ~c1l .
  eq  hyp3 = ~s1g .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  s1g o  s4r)
                                              -- "uses the Lemma6-C-4."
          | FORALL-ACTION (hyp1 o ~s1g)
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t5 o  c1l o  tyl)
                                              -- "uses the lemma6-C-11."
          | FORALL-ACTION (hyp3 o  c1t5 o ~c1l)
          | FORALL-ACTION (hyp3 o ~c1t5) .
}
```

補題 **6.C.14**

```
mod LEMMA6-C-14 {
  ex (PROOF1)
  eq  p (S:State) = (not pos (c1, S) = t4)
          or-else  (not signal (s1, S)  and  not signal (s4, S)) .

  -- base step.
  eq  INI = p (c1tl o c1r)
          | p (c1tr o c1l) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t4 .
  eq  hyp2 = s1r o s4r .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t3 o  c1r o s1r o s4r)
                          -- "uses the Lemma6-A-14 and the Lemma6-C-12."
          | FORALL-ACTION (hyp1 o  c1t3 o ~c1r o s1r)
                          -- "uses the Lemma6-A-14."
          | FORALL-ACTION (hyp1 o  c1t5 o  c1l o s1r o s4r)
                          -- "uses the Lemma6-A-15 and the Lemma6-C-13."
          | FORALL-ACTION (hyp1 o  c1t5 o ~c1l o s4r)
                          -- "uses the Lemma6-A-14."
          | FORALL-ACTION (hyp1 o ~c1t3 o ~c1t5)
          | FORALL-ACTION (hyp2 o  c1t4 o  t4nz)
          | FORALL-ACTION (hyp2 o ~c1t4) .
}
```

補題 **6.C.15**

```
mod LEMMA6-C-15 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t3)
          or-else  (not dir (c1, S) = r)
          or-else  (not pos (c2, S) = t5)
          or-else  (not dir (c2, S) = l) .

  -- base step.
  eq  INI = p (c1tl o c2tr)
          | p (c1tr o c2tr)
          | p (c1tl o c2tl)
          | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 hyp4 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~c1r .
  eq  hyp3 = ~c2t5 .
  eq  hyp4 = ~c2l .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t5 o  c2l  o s1r)
                                              -- "uses the Lemma6-C-13"
          | FORALL-ACTION (hyp1 o  c2t5 o ~c2l)
          | FORALL-ACTION (hyp1 o ~c2t5)
          | FORALL-ACTION (hyp2)
          | FORALL-ACTION (hyp3 o  c1t3 o  c1r  o s4r)
                                              -- "uses the Lemma6-C-12"
          | FORALL-ACTION (hyp3 o  c1t3 o ~c1r)
          | FORALL-ACTION (hyp3 o ~c1t3)
          | FORALL-ACTION (hyp4) .
}
```

補題 **6.C.16**

```
mod LEMMA6-C-16 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t4)
           or-else (not pos (c2, S) = t3)
           or-else (not dir (c2, S) = r) .

  -- base step.
```

```
  eq  INI = p (c1tl o c2tr)
        | p (c1tr o c2tr)
        | p (c1tl o c2tl)
        | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t4 .
  eq  hyp2 = ~c2t3 .
  eq  hyp3 = ~c2r .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t3 o  c2t3)
                                  --> "uses the Claim3, unrechable."
        | FORALL-ACTION (hyp1 o  c1t3 o ~c2t3)
        | FORALL-ACTION (hyp1 o ~c1t3 o  c2t3 o  c2r o  c1t5 o  c1l)
                                  --> "uses the Lemma6-C-15, unrechable."
        | FORALL-ACTION (hyp1 o ~c1t3 o  c2t3 o  c2r o  c1t5 o ~c1l)
        | FORALL-ACTION (hyp1 o ~c1t3 o  c2t3 o  c2r o ~c1t5)
        | FORALL-ACTION (hyp1 o ~c1t3 o  c2t3 o ~c2r)
        | FORALL-ACTION (hyp1 o ~c1t3 o ~c2t3)
        | FORALL-ACTION (hyp2 o  c1t4 o s1r)  -- "uses the Lemma6-C-14."
        | FORALL-ACTION (hyp2 o ~c1t4)
        | FORALL-ACTION (hyp3) .
}
```

補題 **6.C.17**

```
mod LEMMA6-C-17 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t4)
          or-else (not pos (c2, S) = t5)
          or-else (not dir (c2, S) = l) .

  -- base step.
  eq  INI = p (c1tl o c2tr)
        | p (c1tr o c2tr)
        | p (c1tl o c2tl)
        | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 hyp3 : -> Case
  eq  hyp1 = ~c1t4 .
  eq  hyp2 = ~c2t5 .
  eq  hyp3 = ~c2l .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c1t5 o  c2t5)
                                   --> "uses the Claim5, unrechable."
        | FORALL-ACTION (hyp1 o  c1t5 o ~c2t5)
        | FORALL-ACTION (hyp1 o ~c1t5 o  c2t5 o  c2l o  c1t3 o  c1r)
                                   --> "uses the Lemma6-C-15, unrechable."
        | FORALL-ACTION (hyp1 o ~c1t5 o  c2t5 o  c2l o  c1t3 o ~c1r)
        | FORALL-ACTION (hyp1 o ~c1t5 o  c2t5 o  c2l o ~c1t3)
        | FORALL-ACTION (hyp1 o ~c1t5 o  c2t5 o ~c2l)
        | FORALL-ACTION (hyp1 o ~c1t5 o ~c2t5)
        | FORALL-ACTION (hyp2 o  c1t4 o  s4r)  -- "uses the Lemma6-C-14."
        | FORALL-ACTION (hyp2 o ~c1t4)
        | FORALL-ACTION (hyp3) .
}
```

C.2.6　表明 6.1 から 6.7

```
mod CLAIM1 {
  ex (PROOF)
  eq p (S:State) = (not pos (c1, S) = t1)
                   or-else (not pos (c2, S) = t1) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
            | p (c1tr o c2tr)
            | p (c1t1 o c2t1)
            | p (c1tr o c2t1) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t1 .
  eq  hyp2 = ~c2t1 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t1 o  t1nz)  -- "uses the Lemma6-A-1."
            | FORALL-ACTION (hyp1 o ~c2t1)
            | FORALL-ACTION (hyp2 o  c1t1 o  t1nz)  -- "uses the Lemma6-A-1."
            | FORALL-ACTION (hyp2 o ~c1t1) .
}
mod CLAIM2 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t2)
                   or-else (not pos (c2, S) = t2) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
            | p (c1tr o c2tr)
            | p (c1t1 o c2t1)
            | p (c1tr o c2t1) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t2 .
  eq  hyp2 = ~c2t2 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o ~c2t2)
            | FORALL-ACTION (hyp1 o  c2t2 o  c1t3 o  c1l)
                  --> "uses the Lemma6-B-3, unreachable."
            | FORALL-ACTION (hyp1 o  c2t2 o  c1t3 o ~c1l)
            | FORALL-ACTION (hyp1 o  c2t2 o ~c1t3)
            | FORALL-ACTION (hyp2 o ~c1t2)
            | FORALL-ACTION (hyp2 o  c1t2 o  c2t3 o  c2l)
                  --> "uses the Lemma6-B-3, unreachable."
            | FORALL-ACTION (hyp2 o  c1t2 o  c2t3 o ~c2l)
            | FORALL-ACTION (hyp2 o  c1t2 o ~c2t3) .
}
mod CLAIM3 {
ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t3)
                   or-else (not pos (c2, S) = t3) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
            | p (c1tr o c2tr)
            | p (c1t1 o c2t1)
            | p (c1tr o c2t1) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t3 .
  eq  hyp2 = ~c2t3 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t3 o s1r o s2r)  -- "uses the Lemma6-A-14."
            | FORALL-ACTION (hyp1 o ~c2t3)
            | FORALL-ACTION (hyp2 o  c1t3 o s1r o s2r)
                  -- "uses the Lemma6-A-14."
            | FORALL-ACTION (hyp2 o ~c1t3) .
}
mod CLAIM4 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t4)
                   or-else (not pos (c2, S) = t4) .

  -- base step.
  eq  INI = p (c1t1 o c2tr)
            | p (c1tr o c2tr)
            | p (c1t1 o c2t1)
            | p (c1tr o c2t1) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t4 .
  eq  hyp2 = ~c2t4 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t4 o  c1t3 o                  c1r)
                  --> "uses the Lemma6-C-16, unreachable."
            | FORALL-ACTION (hyp1 o  c2t4 o  c1t3 o                 ~c1r)
            | FORALL-ACTION (hyp1 o  c2t4 o                  c1t5 o  c1l)
                  --> "uses the Lemma6-C-17, unreachable."
            | FORALL-ACTION (hyp1 o  c2t4 o                 ~c1t5 o  c1l)
            | FORALL-ACTION (hyp1 o  c2t4 o ~c1t3 o ~c1t5)
            | FORALL-ACTION (hyp1 o ~c2t4)
            | FORALL-ACTION (hyp2 o  c1t4 o  c2t3 o                  c2r)
                  --> "uses the Lemma6-C-16, unreachable."
            | FORALL-ACTION (hyp2 o  c1t4 o  c2t3 o                 ~c2r)
            | FORALL-ACTION (hyp2 o  c1t4 o                  c2t5 o  c2l)
                  --> "uses the Lemma6-C-17, unreachable."
```

```
         | FORALL-ACTION (hyp2 o  c1t4 o          c2t5 o  ~c2l)
         | FORALL-ACTION (hyp2 o  c1t4 o ~c2t3 o ~c2t5)
         | FORALL-ACTION (hyp2 o ~c1t4) .
}

mod CLAIM5 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t5)
           or-else (not pos (c2, S) = t5) .

  -- base step.
  eq  INI = p (c1tl o c2tr)
          | p (c1tr o c2tr)
          | p (c1tl o c2tl)
          | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t5 .
  eq  hyp2 = ~c2t5 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t5 o s3r o s4r)
                                                  -- "uses the Lemma6-A-15."
          | FORALL-ACTION (hyp1 o ~c2t5)
          | FORALL-ACTION (hyp2 o  c1t5 o s3r o s4r)
                                                  -- "uses the Lemma6-A-15."
          | FORALL-ACTION (hyp2 o ~c1t5) .
}

mod CLAIM6 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t6)
           or-else (not pos (c2, S) = t6) .

  -- base step.
  eq  INI = p (c1tl o c2tr)
          | p (c1tr o c2tr)
          | p (c1tl o c2tl)
          | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t6 .
  eq  hyp2 = ~c2t6 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o ~c2t6)
          | FORALL-ACTION (hyp1 o  c2t6 o  c1t5 o  c1r)
                                  --> "uses the Lemma6-B-4, unreachable."
          | FORALL-ACTION (hyp1 o  c2t6 o  c1t5 o ~c1r)
          | FORALL-ACTION (hyp1 o  c2t6 o ~c1t5)
          | FORALL-ACTION (hyp2 o ~c1t6)
```

```
         | FORALL-ACTION (hyp2 o  c1t6 o  c2t5 o  c2r)
                                  --> "uses the Lemma6-B-4, unreachable."
         | FORALL-ACTION (hyp2 o  c1t6 o  c2t5 o ~c2r)
         | FORALL-ACTION (hyp2 o  c1t6 o ~c2t5) .
}

mod CLAIM7 {
  ex (PROOF2)
  eq p (S:State) = (not pos (c1, S) = t7)
           or-else (not pos (c2, S) = t7) .

  -- base step.
  eq  INI = p (c1tl o c2tr)
          | p (c1tr o c2tr)
          | p (c1tl o c2tl)
          | p (c1tr o c2tl) .

  -- induction hypothesis.
  ops hyp1 hyp2 : -> Case
  eq  hyp1 = ~c1t7 .
  eq  hyp2 = ~c2t7 .

  -- induction step.
  eq  IND = FORALL-ACTION (hyp1 o  c2t7 o t7nz)  -- "uses the Lemma6-A-7."
          | FORALL-ACTION (hyp1 o ~c2t7)
          | FORALL-ACTION (hyp2 o  c1t7 o t7nz)  -- "uses the Lemma6-A-7."
          | FORALL-ACTION (hyp2 o ~c1t7) .
}
```