Modeling in Event-B: System and Software Engineering
イベント B でモデリング: システムとソフトウェア工学

Jean-Raymond Abrial

2011 年 9 月 14 日

目次

第 2 章

橋上の自動車の制御

1　序論

この章の目的は小さなシステム開発の完結した例を示すことである。開発の過程で、読者は私たちの体系的な手法を意識するだろう: それは、段階的に正確になるモデルの系列を作るというものである。連続したモデル系列を作る理由は、初めから複雑なモデルを作ってしまうと推論が非常に困難だからである。それぞれのモデルは高級言語によるプログラミングを表現するのではないことを注意しておく。むしろモデルはシステムの***外部観察者*** *(external observer)* からどのように見えるかを定式化する。

それぞれの段階のモデルは解析および証明されることでいくつかの基準に対して***正しい*** *(correct)* という確証が得られる。その結果、最後のモデル (最終モデル) が証明されたとき、このモデルは***構成的に正しい*** *(correct by construction)* と言うことができる。その上、このモデルは最終的な実装にとても近く、本物のプログラムに容易に変換できる。

前述の正しさの基準は、***証明義務規則*** *(proof obligation rule)* をモデルに適用することで、完全に明確かつ体系的にすることができる。この規則を適用すると、私たちはいくつかの言明[*1]を形式的に証明しなければならなくなる。この目的のために、私たちは古典的な***シークエント計算の推論規則*** *(rules of inference of the sequent calculas)* を思い出す必要がある。このような導出規則は命題論理 (propositional logic)、等式 (equality)、初等算術 (basic arithmetic) を含む。これらの観念は、証明義務規則が生成する言明を読者が手動で証明するのを可能にする。明らかに、このような証明は自動定理証明器でも可能だが、それを使う前に手動での証明を練習しておくことをお勧めする。ちなみに、自動証明器の動作はここで説明するものとは異なることを注意しておく。ほとんどの自動証明器は、人間がする証明とは異なる方法で動作する。

この章の構成は以下のようになっている。2 節はシステムの要求文書である。これは第 1 章で説明した原理 (独立した、通し番号を持つ、簡潔な表記) に従う。3 節は精義化戦略を多様な開発工程の多様な要求に対して説明する。続く 4 の節 (4 節から 7 節) は、初期モデルの作成と 3 回の精義化の記述である。

2　要求文書

私たちが作ろうとしているシステムは***コントローラー*** *(controller)* と呼ばれるソフトウェアで、***環境*** *(environment)* と呼ばれる設備に接続されている。そのため、要求には以下の 2 種類がある。コントローラーの機能に関するものと、環境に関するもの。前者はラベル FUN、後者はラベル ENV を付与する。

[*1] 訳注:「言明」は原文では statement。論理学で証明の対象となる文を statement と言い、sentence とは言わない。

これから作ろうとするモデルは***閉じたモデル*** *(closed model)* であり、コントローラーと***環境を同等に含む*** *(as well as its environment)* ことを指摘しておく。その理由は、環境についての前提を最大限に注意深く定義したいからである。別の言い方をすれば、コントローラーが***正しい*** *(correct)* のは環境がこれらの前提に従っている場合に過ぎない。その前提の外では、コントローラーの正しさは保証されない。この問題は 7 節で再考する。

話題を要求文書に戻す。このシステムの主要な機能は、狭い橋の上の自動車を制御することである。この橋は本土と島とを結ぶものとする。

> このシステムは、本土と島を結ぶ橋の上の自動車を制御する。[FUN-1]

このシステムは 2 個の信号機を備えている。

> このシステムは、青と赤の 2 種類の色を表示する 2 個の信号機を備えている。[ENV-1]

信号機のうちの一方は本土に置かれ、もう一方は島に置かれている。どちらも橋のすぐ近くにある。

> 信号機は橋の両端にある入り口をそれぞれ制御する。[ENV-2]

運転手は信号機に従い、信号が赤のときは通過しないことを前提とする。

> 自動車は赤信号を通過せず、青信号のみを通過することを前提とする。[ENV-3]

また、橋の両端には自動車センサーがある。

> このシステムは、オンとオフの 2 種類の状態を取る 4 個のセンサーを備えている。[ENV-4]

これらのセンサーは、橋に入ろうとしている、または橋から出ようとしている自動車の存在を感知する。そのようなセンサーは 4 個ある。そのうちの 2 個は橋の上に置かれており、残りの 2 個はそれぞれ本土と島に置かれている。

> センサーは橋に入るまたは橋から出る自動車の存在を検出する。「オン」は自動車が橋に入るまたは橋から出ようとしていることを意味する。[ENV-5]

これまで説明した設備の構成要素は図 1[*2]に示されている。このシステムには他に重要な制約がある。橋の上および島の自動車の数は制限されている。

> 橋の上および島の自動車の数には限界がある。[FUN-2]

橋は片側通行である。

> 橋は片側通行であり、両方向に同時に走行できない。[FUN-3]

[*2] 訳注: 訳文では省略。

3　精義化の戦略

このシステムの開発に取り掛かる前に、私たちのデザインの戦略を明確に定義しておくことが有益である。そのために、前節の要求文書で挙げたさまざまな要求の、考慮する順番をリストにして示す。

- まず、要求 FUN-2 のみを考慮する単純なモデルから始める (4 節)。FUN-2 は橋および島の自動車の数の制限である。
- 次に、橋を図に書き入れ、橋が片側通行であること [FUN-3] を考えに入れる (5 節)。
- 続いて、信号機を導入する。これは要求 ENV-1, ENV-2, ENV-3 に対応する (6 節)。
- 最後の精義化ではセンサー [ENV-4, ENV-5] を導入する (7 節)。さらに、この精義化で、コントローラー、環境およびそれらの通信チャネルから成る閉じたモデルの**アーキテクチャー** *(architecture)* を導入する。

ここで要求 FUN-1——システム全体の機能が何であるかを教えている——について述べていないことに気が付くかもしれない。要求 FUN-1 はシステム全般についての規定であり、全ての開発ステップで考慮に入れる。

4　初期モデル: 自動車の数の制限

4.1　序論

私たちが構築する初期モデルは非常に単純である。信号機やセンサーのような多数の構成要素については考えず、それらは後の精義化で導入する。さらに、橋についても考えず、橋と島の***合成*** *(compound)* についてのみ扱う。

これは非常によく使う考え方である。私たちは、最終的に作ろうとしているシステムよりも***はるかに抽象的な*** *(far more abstruct)* モデルから始める。そのため、最初は極めて少数の制約のみを考える。これは、それぞれの要求をその都度考えることで、システムについての推論をシンプルにするためである。

有益な比喩として、空高くから見下ろす状況を想像できる。橋は見えず、「島と橋」の上の自動車が見え、自動車が「島と橋」に出たり入ったりすることによる ML_out と ML_in という 2 種類の遷移が観察できる。これらの様子を図 2 に示す。

私たちの最初の仕事は、このシンプルなモデルの***状態*** *(state)* を定式化することである。(4.2 節) その後、2 種類の***イベント*** *(events)* を定式化する (4.3 節)

4.2　状態の定式化

モデルの状態は、***静的部分*** *(static part)* と***動的部分*** *(dynamic part)* で構成される。静的部分は***定数*** *(constant)* の定義と公理で構成される。対照的に、動的部分はシステムの変化によって書き換えられる***変数*** *(variable)* で構成される。静的部分はモデルの***コンテキスト*** *(context)* とも言う。

私たちの初期モデルのコンテキストはとても単純である。「島と橋」に同時に存在できる自動車の数の最大値を表す定数 d だけが含まれる。定数 d は単純な公理自然数であるのみを持つ。この公理の名前は **axm0_1** とする。

定数: d　　**axm0_1**: $d \in \mathbb{N}$

動的部分は、ある時刻での「橋と島」にある自動車の数を表す変数 n から成る。これは以下のように書ける。

変数: n　　**inv0_1**: $n \in \mathbb{N}$
inv0_2: $n \leq d$

変数 n は***不変式*** *(invariant)* と呼ばれる 2 個の条件で定義される。その名前は **inv0_1** と **inv0_2** である。「不変式」という用語の由来は、変数 n の値が変わっても、不変式は常に真だからである。不変式 **inv0_1** は n が自然数であることを表す。また、不変式 **inv0_2** は、「島と橋」の自動車の数が最大値 n よりも常に小さいか等しいことを表す。***最初の基本的な要求*** *(first basic requirement)* である FUN-2 は、この段階で不変式 **inv0_2** を宣言することによって考慮に入れられる。

axm0_1, **inv0_1**, **inv0_2** のようなラベルは体系的に命名されている。接頭辞 **axm** は***公理*** *(axiom)* を表し、**inv** は***不変式*** *(invariant)* を表す。数字 0 は初期モデルで導入されたことを表す。それ以降の精義化されたモデルは番号を増やす。最後に、**inv0_2** の 2 のような数字は単なる連番である。以降このような体系的な命名規則を

状態の条件のために使用する。稀に **axm** や **inv** のような接頭辞を変更することはある。この命名規則は便利であるが、もちろん、別の体系的な命名規則もあり得る。

4.3 イベントの定式化

この段階では、以降***イベント*** *(events)* と呼ぶ 2 種類の遷移が観察できる。これらは自動車が「島と橋」に出たり入ったりすることを表す。イベント ML_out が発生する直前と直後の様子が図 3 に示されている。ML_out という名前は、本土から出ることを表す。明らかに、このイベントによって、「島と橋」の自動車の数が増加する。

同様に、図 4 はイベント ML_in の直前と直後の様子である。このイベントによって「島と橋」の自動車の数は減少する。

第 1 ***の近似*** *(first approximation)* として、初期モデルのイベントを以下のように定義する。

ML_out	ML_in
$n := n + 1$	$n := n - 1$

イベントは ML_in, ML_out のような***名前*** *(name)* を持つ。また、イベントは $n := n + 1$ や $n := n - 1$ のような***動作*** *(action)* を持つ。これらの言明は「n は $n + 1$ に等しくなる」「n は $n - 1$ に等しくなる」と読める。これらの言明が***動作*** *(actions)* である。重要な注意点は、これらの動作を書くことは***プログラミングをすることではない*** *(we are not programming)* ということである。離散時間でのシステムの変化に伴って観察できるものを、私たちは形式的に表現する。私たちは観察したものの形式な表現を与える。

これらのイベントは「第 1 の近似」でしかないことに気が付くかもしれない。なぜなら、

1. 私たちのモデルの解析は***漸進的*** *(incremental fashion)* である。すなわち、システムの最終的な状態とイベントを一気に定義することはない。***私たちはプログラミングをせず*** *(we are not programming)*、システムのモデルを定義していること、全てを網羅して一度に定義するわけではないことを思い出す必要がある。そのため、状態の要素と遷移は少しずつ段階的に導入される。
2. 私たちは状態とイベントを導入したが、それらが無矛盾であることを確かめていない。このことは形式的に証明しないといけない。さらに、証明の過程で、現時点での状態とイベントが正しくないことが判明してしまう。

4.4 前後述語

この節では***前後述語*** *(before-after predicate)* の概念を説明する。この概念は後の節で証明義務規則を説明するのに役に立つ。

動作によって定義されているイベントは、ここでは前後述語に対応する。前後述語は一つの動作に結びつけられており、注目している変数の、遷移の***直前*** *(just before)* と***直後*** *(just after)* の値の関係を表現している。

イベント	ML_out $n := n + 1$	ML_in $n := n - 1$
前後述語	$n' = n + 1$	$n' = n - 1$

すぐ分かるように、前後述語は動作を以下のように変換して得られる。左辺の変数にプライムを付け、記号

:= を = に変え、右辺はそのままにする。

前後述語では、***約束事として*** *(by convention)*、n' のようなプライム付き変数は変数 n の遷移***直後*** *(just after)* の値であり、対照的に、n は遷移***直前*** *(just before)* の値である。簡単に言えば、イベント ML_out の直後には、変数 n の値はイベント直前の値に 1 を足したものであり、それが $n' = n + 1$ である。

ここで説明した前後述語は、プライム付き変数がプライム無し変数に依存する何らかの式に等しいという、単純な形をしている。この例は***決定的*** *(deterministic)* であるが、もっと複雑な例も後に現れる。

4.5 不変式の保存の証明

イベント ML_out および ML_in の動作を定義したときには、不変式 **inv0_1** および **inv0_2** を考慮しなかった。なぜなら、イベントの定義では変数 n がどのように変化するかを考えればよいからである。そのため、これらのイベントによって不変式が保存されるかどうかは証明しなければ分からない。実際のところ、***これは厳格に証明する必要がある*** *(it has to be proved in a rigorous fashion)*。この節の目的は、不変式が本当に「不変」であることを保証するために証明しなくてはならない言明を詳細に定義することである。

証明が必要な言明は INV という名前の規則によって体系的に生成される。規則 INV は一度定義すれば何にでも使える。これに類する規則は***証明義務*** *(proof obligation)* 規則または***検証条件*** *(verification condition)* 規則と言う。

一般に、定数は集合的に c と書くものとする。$A(c)$ は定数 c についての公理である。詳しく言うと、$A(c)$ は定数 c に関する公理のリスト $A_1(c), A_2(c), ...$ を表す。私たちの例では、$A(c)$ はただ一つの要素を持つリストであり、その要素は公理 **axm0_1** である。同様に、v は変数を表し、$I(c, v)$ はその変数についての不変式を表す。また、$I(c, v)$ は不変式のリスト $I_1(c, v), I_2(c, v), ...$ を表す。私たちの例では、$I(c, v)$ は 2 個の要素 **inv0_1** および **inv0_2** から成るリストである。最後に、イベントの前後述語を $v' = E(c, v)$ とする。特定のイベントと、不変式の集合 $I(c, v)$ から選ばれた不変式 $I_i(c, v)$ に対応して、証明を必要とする不変式保存言明は以下のようになる。

$$A(c), I(c, v) \vdash I_i(c, E(c, v)) \text{ [INV]}$$

この言明は、***シークエント***[*3] *(sequent)* と呼ばれ、「仮説 $A(c)$ と仮説 $I(c, v)$ は述語 $I_i(c, E(c, v))$ を***導く*** *(entail)*」と読む。これは全てのイベントおよび全ての不変式 $I_i(c, v)$ それぞれに対して証明しなくてはならない。これを理解するのは易しい。遷移の前には、明らかに全ての公理 $A(c)$ が正しいことを前提にできる。同様に全ての不変式 $I(c, v)$ が正しいことも前提にできる。その結果、$A(c)$ と $I(c, v)$ を前提にできる。ところで、遷移の直後には、変数 v の値は $E(c, v)$ に変化し、不変式の言明 $I_i(c, v)$ は $I_i(c, E(c, v))$ に変化する。そして、***不変式*** *(invariant)* である以上は、この新しい不変式も正しくなくてはならない。

読み書きを容易にするために、仮説が複数あるときには、私たちはシークエントを縦に並べて書く。規則 INV の例は以下のようになる。

$$\begin{array}{l} A(c) \\ I(c, v) \\ \vdash \\ I_i(c, E(c, v)) \text{ [INV]} \end{array}$$

[*3] シークエントの詳細な定義は次の節まで待ってほしい。

証明義務規則 INV のこのような定式化は覚えにくいので、少しくだけた表現を示す。

公理
不変式
⊢
変更後の不変式 [INV]

証明義務規則 INV は、さまざまなイベントが不変式を維持することを保証するために***形式的に証明する必要がある言明*** *(what we have to formally prove)* を教えてくれる。しかし私たちはまだ「形式的に証明する」とは何か定義していない。それは 4.8 節から 4.11 節で定義する。さらに私たちは体系的な形式的証明を組み立てる方法を明らかにしなければならない。最後に、規則 INV を適用することで得られるシークエントは、***証明義務生成器*** *(Proof Obligation Generator)* と呼ばれるツールで簡単に生成できることを注意しておく。

4.6 シークエント

前節では、証明義務規則の概念を説明するために、シークエントの概念を導入した。この節では、このような構造についてもう少しだけ説明する[*4]。すでに説明したように、以下に挙げるような形式の言明を***シークエント*** *(sequent)* と呼ぶ。

$$\mathbf{H} \vdash \mathbf{G}$$

記号 ⊢ は***回転扉*** *(turnstile)* という名前である。回転扉の左辺ここでは **H** は***仮説*** *(hypotheses)* または***前提*** *(assumptions)* と呼ばれ、述語の有限集合である。この集合は空でもよい。回転扉の右辺ここでは **G** は***ゴール*** *(goal)* または***結論*** *(conclusion)* と呼ばれる述語である。

このような言明の直観的な意味は、結論 **G** は前提の集合 **H** ***のもとで*** *(under)* 証明可能であるということである。言い換えれば、回転扉は動詞「導く」または「生成する」と読める。

続く節で、私たちは何度もこのようなシークエントを生成し、証明しようと試みる。また、シークエントを形式的に証明するための規則を説明する。

4.7 不変式保存規則を適用する

私たちの例に戻り、私たちが証明すべきものをはっきりさせる。それは私たちがこの節ですべきことである。証明義務規則 INV は証明すべきいくつかのシークエントを生成する。この規則はイベントごと、そして不定式ごとに適用する。私たちの例には、2 個のイベント (ML_out, ML_in) と 2 個の不変式 (**inv0_1**, **inv0_2**) があるため、計 4 個のシークエントを証明する必要がある。

どの証明義務について話しているのか分かりやすくするため、それらに***合成された名前*** *(compound names)* を付けることにする。まずイベントの名前、次に不変式の名前、最後に不変式保存規則のラベル INV を繋げたものを証明義務の名前とする[*5]。私たちの例では、この方法で命名した 4 個の証明義務は以下のようになる。

[*4] シークエントおよびシークエント計算については 9 章 1 節でより形式的に説明する。
[*5] 別の種類の証明義務も存在する。全ての証明義務は第 5 章の 2 節で見ることができる。

ML_out / **inv0_1** / INV
ML_out / **inv0_2** / INV
ML_in / **inv0_1** / INV
ML_in / **inv0_2** / INV

それでは、2 個のイベントと 2 個の不変式に対して証明義務規則 INV を適用してみよう。まず、イベント ML_out と不変式 **inv0_1** について証明すべきものは、

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq d$	ML_out / **inv0_1** / INV
⊢	⊢	
変更後の不変式 **inv0_1**	$n + 1 \in \mathbb{N}$	

イベント ML_out は前後述語 $n' = n + 1$ を持っている。そのため、不定式 **inv0_1** が前提としては $n \in N$ であるのに対して、結論では $n + 1 \in N$ に書き換えられている。さらに、イベント ML_out と不変式 **inv0_2** について証明すべきものを示す。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq \mathbb{N}$	ML_out / **inv0_2** / INV
⊢	⊢	
変更後の不変式 **inv0_2**	$n + 1 \leq d$	

イベント ML_in と不変式 **inv0_1** について証明すべきものを以下に示す。ここで、ML_in の前後述語が $n' = n - 1$ であることを思い出しておく。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$b \leq d$	ML_in / **inv0_1** / INV
⊢	⊢	
変更後の不変式 **inv0_1**	$n - 1 \in \mathbb{N}$	

イベント ML_in と不定式 **inv0_2** について証明すべきものは以下。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq d$	ML_in / **inv0_2** / INV
⊢	⊢	
変更後の不変式 **inv0_2**	$n - 1 \leq d$	

4.8 証明義務を証明する

私たちは証明すべきシークエントを正確に知ることができた。次の課題はこれを証明することである。これが本節の目的である。これらのシークエントの形式的な証明は、シークエントにいくつかの***変形***

(transformation) を行い、別のシークエントを得て、それを明らかに証明されたと思われるまで繰り返すことで達成される。あるシークエントを別のシークエントに変形することは、後者を証明することで前者も証明されたことになるという意味に相当する。例えば、私たちの最初のシークエント

$$
\begin{array}{l}
\quad d \in \mathbb{N} \\
\quad n \in \mathbb{N} \\
\quad n \leq d \\
\vdash \\
\quad n + 1 \in \mathbb{N} \quad ...(1)
\end{array}
$$

は、***無関係な仮説*** *(irrelevant hypotheses)* を取り除くことによって (明らかに、仮説 $d \in \mathbb{N}$ と仮説 $n \leq d$ は結論 $n + 1 \in N$ の証明には無意味である) 以下の単純なシークエントを得る。

$$n \in \mathbb{N} \vdash n + 1 \in \mathbb{N} \quad ...(2)$$

この過程で私たちが認めていることは、シークエント (2) を証明することはシークエント (1) の証明を満足する (sufficient) という事実である。言い換えれば、シークエント (2) を証明すれば、シークエント (1) も証明したことになる。シークエント (2) の証明は「無」にまで減量できる。これは、n が自然数ならば $n + 1$ も自然数であることを暗黙の前提としているからである。

4.9 推論規則

前節では、シークエントを別のシークエントに変形する規則と、他の正当化を必要としないシークエントを受け入れる規則を非公式に導入した。これらの規則を厳格に形式化したものを***推論規則*** *(rules of inference)* と呼ぶ。私たちの最初の推論規則を以下に示す。

H1 ⊢ **G**
—MON—
H1, H2 ⊢ **G**

これが推論規則の構造である。水平線の上にシークエントの集合が置かれている (この例では 1 個だけである)。これらのシークエントは推論規則の***前件*** *(antecedents)* と呼ばれる。水平線の下には***後件*** *(consequent)* と呼ばれる 1 個のシークエントがある。規則の右 (訳注: 訳文では水平線の中央) には MON と書いてある。これは推論規則の名前である。この名前は仮説の***単調性*** *(monotonicity)* に由来する。

この推論規則は次のように読める。「この後件を証明するには、全ての前件のシークエントが証明されればよい」ここで挙げた例では、2 個の仮説の集合 **H1** および **H2** から結論 **G** を証明するためには、**H1** のみから **G** が証明されればよい。ここで、私たちにが本当に望む効果は、無関係な仮説 **H2** を除去することである。

この推論規則を適用するとき、仮説の集合 **H2** は仮説の集合 **H1** の後に置かれている必要はない。部分集合 **H2** は仮説の任意の部分集合であると考えてよい。例えば、前節で証明義務 (1) にこの推論規則を適用したとき、私たちは仮説 $d \in \mathbb{N}$ と $n \leq d$ を除去した。このとき、除去されなかった仮説 $n \in \mathbb{N}$ を含めて、仮説は $d \in \mathbb{N}, n \in \mathbb{N}, n \leq d$ の順に並んでいた。

2 番目の推論規則はこのように書ける。

—P2—
$\mathbf{H}, \mathbf{n} \in \mathbb{N} \vdash \mathbf{n} + 1 \in \mathbb{N}$

この推論規則は前件がない。そのため、これは**公理**[*6] *(axiom)* と呼ばれる。これは自然数についてのペアノの第 2 公理である。名前 P2 はそれにちなんでいる。この推論規則は、後件を証明するために何も証明する必要がないことを表している。**n** が自然数であるという前提のもとで、**n** + 1 は常に自然数である。追加の仮説 **H** の存在は任意であることを注意しておく。なぜなら、追加の仮説 **H** は推論規則 MON で除去できるからである。そのため、この推論規則は以下のように単純化できる[*7]。

—P2—
$\mathbf{n} \in \mathbb{N} \vdash \mathbf{n} + 1 \in \mathbb{N}$

似ているがより制限された規則を後の節で使用する。これは自然数の減小についての規則である。

—P2'—
$0 < n \vdash n - 1 \in \mathbb{N}$

(訳注: 0 と比較可能であることが **n** が自然数であることを暗示しているらしい) この規則は **n** が正であるという前提のもとで **n** − 1 が自然数であることを意味している。さらに INC と DEC という 2 種類の規則を導入する。

—INC—
$\mathbf{n} < \mathbf{m} \vdash \mathbf{n} + 1 \leq \mathbf{m}$

推論規則 INC は、**n** が **m** より小さいという前提のもとでは **n** + 1 は **m** よりも小さいか等しいことを意味する。

—DEC—
$\mathbf{n} \leq \mathbf{m} \vdash \mathbf{n} - 1 < \mathbf{m}$

推論規則 DEC は、**n** が **m** よりも小さいか等しいとき **n** − 1 が **m** よりも小さいことを意味する。論理や自然数を扱うためには明らかにもっと多くの規則が必要であるが、この節ではこれだけしか使用しないものとする。

推論規則 P2',INC, DEC は***派生*** *(derived)* 規則であることを注意しておく。これは単純に、より基本的な他の規則から演繹することができるという意味である。しかしながら、本書ではこのことはあまり興味を引かない。私たちは単に、推論規則の使いやすい***ライブラリ*** *(library)* を作りたいからである。

4.10 メタ変数

前節で説明した推論規則では、**H1**, **H2**, **G**, **n**, **m** などの太字で表記された識別子が使われていた。これは、これらの変数が数学言語における変数ではなく、***メタ変数*** *(meta-variables)* だからである。

詳しく言うと、それぞれの推論規則は、明示することができない全ての可能な***置き換え*** *(match)* が成された規則を表すスキーマである。例えば、規則 P2

[*6] 訳注: 定数に関する述語も axiom と呼んでいますが、別のものです。
[*7] 後の節ではこちらの約束事に従う。

—P2—
$\mathbf{n} \in \mathbb{N} \vdash \mathbf{n} + 1 \in \mathbb{N}$

はペアノの第 2 公理を、極めて総称的な用語で表現している。これは以下のようなシークエントにも適用できる。

$a + b \in \mathbb{N} \vdash a + b + 1 \in \mathbb{N}$

なぜなら、これはメタ変数 **n** を数学言語の式 $a + b$ に**マッチさせた** *(matching)* 結果だからである。

4.11 証明

数々の推論規則を得たことで、私たちは私たちはいよいよ初歩的な形式的証明に取り掛かる。これがこの節の目的である。証明とはシークエントの列に過ぎない。シークエントは推論規則の名前で繋がれている。シークエントの列は前件を持たない推論規則の名前で終わる。4.24 節では、より普遍化した形の証明を説明するが、ここではこの方法で十分である。

例えば、証明義務 ML_out / **inv0_1** / INV は以下のようになる。これは 4.8 節で非公式に説明したことを正確に表現している。具体的には、不要な仮説を取り除き、2 番目のシークエントを最終的に受け入れている[*8]。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
$\vdash$
$n + 1 \in \mathbb{N}$
—MON—
$n \in \mathbb{N}$
$\vdash$
$n + 1 \in \mathbb{N}$
—P2—

次の証明は証明義務 ML_out / **inv0_2** / INV である。これは推論規則 INC を適用できないために失敗する。最後のシークエントには推論規則 INC が要求する仮説 $n < d$ が含まれておらず、かわりに弱い仮説 $n \leq d$ が含まれている。この理由から、シークエントの列の最後にはクエスチョンマークが置かれている。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
$\vdash$
$n + 1 \leq d$
—MON—
$n \leq d$
$\vdash$
$n + 1 \leq d$
—?—

同様に、証明義務 ML_in / **inv0_1** / INV の証明も失敗する。最後のシークエントが必要な仮説 $0 < n$ ではな

[*8] 訳注: レイアウトの都合により、推論規則の上下が、定義時と使用時で逆になっています。原文では使用時は左右に並んでいます。

く弱い仮説 $n \in \mathbb{N}$ しか持たないため、推論規則 P2' を適用できない。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
⊢
$n - 1 \in \mathbb{N}$
—MON—
$n \in \mathbb{N}$
⊢
$n - 1 \in \mathbb{N}$
—?—

最後の証明 ML_in / **inv0_2** / INV は成功する。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
⊢
$n - 1 \leq d$
—MON—
$n \leq d$
⊢
$n - 1 < d \vee n - 1 = d$
—OR_R1—
$n \leq d$
⊢
$n - 1 < d$
—DEC—

2 番目のステップでは、$n - 1 \leq d$ は $n - 1$ が d より小さい***または*** *(or)* d に等しいことを意味するので、同値な形式的言明 $n - 1 < d \vee n - 1 = 0$ に置き換えている。ここで、記号 $\vee$ は論理和「または」である。そして、推論規則 OR_R1 (次節で説明) を用いて結論 $n - 1 < d$ を得て、最後に推論規則 DEC を適用している。

4.12 その他の推論規則

前節の終わりに、私たちは OR_R1 という名前の、論理に関する最初の推論規則を使った[*9]。後の節ではさらに多くの推論規則を説明するが、OR_R1 はたまたま最初に必要になったものである。この推論規則を OR_R2 とともに示す。

$\mathbf{H} \vdash \mathbf{P}$
—OR_R1—
$\mathbf{H} \vdash \mathbf{P} \vee \mathbf{Q}$

$\mathbf{H} \vdash \mathbf{Q}$
—OR_R2—
$\mathbf{H} \vdash \mathbf{P} \vee \mathbf{Q}$

[*9] 全ての推論規則は第 9 章に挙げる。

これらの推論規則は、2 個の述語 $\mathbf{P}, \mathbf{Q}$ の論理和 $\mathbf{P} \vee \mathbf{Q}$ を結論とする証明についての明らかな (obvious) 事実を表す。論理和を証明するには、論理和を構成するいずれかの述語を証明すれば十分である。

接尾語 R1 および R2 に含まれる文字 R は「右」に由来する。この推論規則は回転扉 $\vdash$ の***右*** *(right)* 辺に置かれた言明、すなわち結論を変形するからである。別の推論規則は接尾語 L を持つことがあり、それは仮説すなわち回転扉の***左*** *(left)* 辺を変形するものである。

4.13 イベントの改良: 限定条件の導入

私たちの例に戻ると、いくつかの証明が失敗したために、モデルの改変が必要になっている。証明はデバッグと同じ効果があることが分かる。なぜなら、***証明の失敗はバグの存在を明らかにする*** *(a failed proof reveals a bug)* からである。

証明の過程で見つかった欠陥を直すために、イベントに***限定条件*** *(guards)* を挿入する必要がある。限定条件はイベントが有効になる***必要条件*** *(necessary condition)* を表現する。詳しく言うと、イベントが有効になると、そのイベントによる遷移を発生させることができる。対照的に、イベントが有効でないとき、——いずれかの限定条件が真でないとき——そのイベントによる遷移は起こらない。

イベント ML_out が有効になるには、$n < d$ すなわち n が d より狭義に小さくなければならない。また、イベント ML_in が有効になるには、$0 < n$ すなわち n が狭義に正でなくてはならない。これらの限定条件は前節のシークエントで証明できなかった条件式とまったく同一であることを注意しておく。私たちは証明の失敗に教えられている。限定条件を挿入したイベントは以下のようになる。

ML_out
when
  $n < d$
then
  $n := n + 1$
end

ML_in
when
  $0 < n$
then
  $n := n - 1$
end

見て分かるように、構文は簡単である。限定条件は用語 **when** と **then** の間にあり、動作は用語 **then** と **end** の間にある。イベントに複数の限定条件を入れることもできるが、この例では 1 個しか入れていない。

4.14 不変式保存規則の改良

限定条件の集合 $G(c, v)$ と前後述語 $v' = E(c, v)$ を持つイベントについての証明義務規則 INV は、シークエントの仮説に限定条件の集合 $G(c, v)$ を加えるように修正される。(定数 c と変数 v の記法は 4.5 節と同じである。) すると、以下のような、限定条件を持つイベントにも適用できる、より普遍化された証明義務が得られる。

$A(c)$ 公理
$I(c, v)$ 不変式
$G(c, v)$ 当該イベントの限定条件
$\vdash$
$I_i(c, E(c, v))$ [INV] 変更後の不変式 [INV]

4.15 不変式の保存を再び証明する

修正された証明義務規則 INV により、証明すべき言明も変化したが、これは簡単に証明できる。以下はイベント ML_out が不変式 inv0_2 を保存することを保証するための証明である。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq d$	ML_out / **inv0_2** / INV
ML_out の限定条件	$n < d$	
$\vdash$	$\vdash$	
変更後の不変式 **inv0_2**	$n + 1 \leq d$	

以下はイベント ML_in と不変式 inv0_1 についてのものである。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq d$	ML_in / **inv0_1** / INV
ML_in の限定条件	$0 < n$	
$\vdash$	$\vdash$	
変更後の不変式 **inv0_1**	$n - 1 \leq d$	

これらの証明は以下のように簡単である。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
$n < d$
$\vdash$
$n + 1 \leq d$
—MON—
$n < d$
$\vdash$
$n + 1 \leq d$
—INC—

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
$n < d$
$\vdash$
$n - 1 \leq d$
—MON—
$0 < n$
$\vdash$
$n - 1 \in \mathbb{N}$
—P2'—

既に 4.11 節で証明した証明義務 ML_out / **inv0_1** / INV と ML_in / **inv0_2** / INV は、ここでやり直す必要はない。なぜなら、ここでしていることは証明義務に仮説を追加するだけであり、推論規則 MON (4.9 節) によれば、**少ない** *(less)* 仮説から結論が得られるならば、さらに仮説を追加しても同じ結論が得られるからである。逆に言えば、既に証明されているシークエントにさらに仮説を追加しても、その証明は正しい。

4.16 初期化

これまでにイベント ML_in, ML_out と不変式 **inv0_1**, **inv0_2** を定義した。また、これらのイベントによる遷移で不変式が保存されることを証明した。しかし、最初の時点で何が起こるかは分からない。そのため、初期

化のための特別なイベントを定義する必要がある。名前は init とする。私たちの例では、このイベントは以下のようになる。

init
$n := 0$

見て分かるように、初期化イベントは、最初は「島と橋」に自動車がいないという観察に対応している。このイベントには限定条件が存在しないことを注意しておく。この例に限らず、初期化イベント init には限定条件が存在しない。初期化イベントは必ず発生しなければならないからである。

さらに、初期化イベントの動作では、記号 := の右辺は変数を参照してはならないことを重ねて注意しておく。これは***初期化*** *(initializing)* という目的に照らし合わせれば自明である。すると、イベント init の前後述語は言ってみれば「後述語」である。後述語は以下のようになる。見て分かるように、この述語は変数 n' を含むが n は含まない。

$n' = 0$

4.17 初期化イベント init のための不変式成立規則

初期化イベント init は不変式を保存することがない。なぜなら初期化イベントの前にはシステムの状態が「存在しない」からである。かわりに初期化イベントは**不変式を成立** *(establish the invariant)* させなければならない。それによって、他の初期化後に観察できるイベントは、不定式が成立した状況で発生することになる。

このような理由により、不変式の成立のための証明義務規則が必要になる。これは 4.5 節で説明した証明義務規則 INV とほとんど同じである。違いは、この節で説明する証明義務規則では、不変式がシークエントの仮説に入らないことである。詳しく言うと、システムに定数 c と公理の集合 $A(c)$ があり、変数 v と不変式 $I(c, v)$ があり、後述語 $v' = K(c)$ を持つ初期化イベントがあるとき、不変式の成立のための証明義務規則 INV は以下のようになる。

$A(c)$ ⊢ $I_i(c, K(c))$ [INV]　　公理 ⊢ 変更後の不変式 [INV]

4.18 不変式成立規則の適用

初期化イベントに不変式成立規則を適応すると、以下のシークエントが得られる。

公理 **axm0_1** ⊢ 変更後の不変式 **inv0_1**　　$d \in \mathbb{N}$ ⊢ $0 \in \mathbb{N}$　　**inv0_1** / INV

続いて

公理 **axm0_1** ⊢ 変更後の不変式 **inv0_2**　　$d \in \mathbb{N}$ ⊢ $0 \leq d$　　**inv0_2** / INV

4.19 初期化のための証明義務を証明する: 新しい推論規則

前節の証明義務は、新たに推論規則を増やさなければ証明できない。新たな推論規則の一つは P1 という。これはペアノの第 1 公理であり、0 が自然数であることを主張している。この推論規則の後件は仮説がないことを注意しておく。

—P1—
$\vdash 0 \in \mathbb{N}$

2 番目の推論規則は後件としてペアノの第 3 公理を持つ。これは、0 はどの自然数よりも大きくないということである。これは 0 が最小の自然数であることを変形して得られる。この推論規則の名前は P3 とする。

—P3—
$n \in \mathbb{N} \vdash 0 \leq n$

2 個の初期化のための証明義務を証明するのは練習問題とする。

4.20 無デッドロック

私たちのイベント ML_in と ML_out は保護されている。そのため、どちらの限定条件も偽であるときモデルは***デッドロック*** *(deadlock)* する。どのイベントも有効にならず、システムは停止させられる。場合によってはこれが望ましい動作であるが、この例ではそうではない。ここで、停止しないという性質は第 2 章の要求文書に欠けていることが判明する。そこで、私たちは要求文書を修正して、以下の要求を追加する。

システムは一度開始したら永久に動き続ける。[FUN-4]

4.21 無デッドロック規則

定数 c、公理の集合 $A(c)$、変数 v、不変式 $I(c,v)$ から成るモデルを考える。証明義務規則 DLF は、限定条件 $G_1(c,v), G_2(c,v), ..., G_m(c,v)$ のうちいずれかが常に真であるという証明義務を生成する。これは、私たちの例では、自動車は常に「島と橋」に入ることができるか出ることができるということである。この証明義務は定数 c についての公理の集合 $A(c)$ と不変式の集合 $I(c,v)$ を前提として証明する。この証明義務規則は、総称的な用語で以下のように言明できる。

$A(c)$	公理
$I(c,v)$	不変式
$\vdash$	$\vdash$
$G_1(c,v) \vee ... \vee G_m(c,v)$ [DLF]	限定条件の論理和 [DLF]

この規則の適用が***常に必要とは限らない*** *(not mandatory)*。無デッドロック性を要求しないシステムもある。

4.22 無デッドロック証明義務規則を適用する

これが証明義務規則 DLF によって生成された証明すべきシークエントである。

公理 **axm0_1**	$d \in \mathbb{N}$	
不変式 **inv0_1**	$n \in \mathbb{N}$	
不変式 **inv0_2**	$n \leq d$	DLF
⊢	⊢	
限定条件の論理和	$n < d \lor 0 < n$	

4.23 新しい推論規則

前掲の無デッドロック証明義務は新たな推論規則がないと証明できない。新たに導入する最初の推論規則は、***場合分けによる証明*** *(proof ob cases)* という古典的な手法である。OR_R という名前は、シークエントの仮説すなわち「左」辺に置かれている論理和 ∨ についての規則だからである。この規則の前件は 2 個のシークエントになることを注意しておく。詳しく言うと、論理和を持つ仮説 $\mathbf{P} \lor \mathbf{Q}$ のもとで結論を証明するには、仮説 **P** からおよび仮説 **Q** から同じ結論を***独立に証明*** *(prove independently)* すればよい。

$\mathbf{H}, \mathbf{P} \vdash \mathbf{R} \quad \mathbf{H}, \mathbf{Q} \vdash \mathbf{R}$
—OR_L—
$\mathbf{H}, \mathbf{P} \lor \mathbf{Q} \vdash \mathbf{R}$

4.12 節で既に紹介した推論規則も再掲する。

$\mathbf{H} \vdash \mathbf{P}$
—OR_R1—
$\mathbf{H} \vdash \mathbf{P} \lor \mathbf{Q}$

$\mathbf{H} \vdash \mathbf{Q}$
—OR_R2—
$\mathbf{H} \vdash \mathbf{P} \lor \mathbf{Q}$

論理についての最後の規則は、シークエントの本質に関わるものである。まず、規則 HYP は、シークエントの結論が仮説として書かれているならばシークエントは証明されたというものである。次に、FALSE_L は、偽の仮説を持つシークエントは証明されたというものである。私たちは偽は ⊥ と書くものとする。

—FALSE_L—
$\bot \vdash \mathbf{P}$

次の 2 個の推論規則は等式を扱う。これらの規則は仮説が等式であるときそれを除去する方法を表す。

$\mathbf{H}(\mathbf{F}), \mathbf{E} = \mathbf{F} \vdash \mathbf{P}(\mathbf{F})$
—EQ_LR—
$\mathbf{H}(\mathbf{E}), \mathbf{E} = \mathbf{F} \vdash \mathbf{P}(\mathbf{E})$

$\mathbf{H}(\mathbf{E}), \mathbf{E} = \mathbf{F} \vdash \mathbf{P}(\mathbf{E})$
—EQ_RL—
$\mathbf{H}(\mathbf{F}), \mathbf{E} = \mathbf{F} \vdash \mathbf{P}(\mathbf{F})$

規則 EQ_LR では、シークエントの結論 **P(E)** は式 **E** に***依存する*** *(depending)* 述語である。仮説 **H(E)** は式 **E** に依存している。最後に、仮説は等式 **E** = **F** を含んでいる。この規則によれば、このシークエントを **P(E)** と **H(E)** に現れる全ての **E** を **F** に置き換えたシークエントに置き換えることができる。LR という名前は等式を左から右に適用することに由来する。規則 EQ_RL は同じ等式を右から左に適用する。すなわち、**P(F)** と

H(F) に含まれる **F** を **E** に置き換えて **P(E)** と **H(E)** を得る。私たちは述語が式 **E** に「依存する」という概念を明確に定義していないことを注意しておく。この件については後ほど詳しく説明する。

等式を扱う最後の規則は、式 **E** はそれ自身に等しいというものである。この規則は次の節での証明には用いないが、ここで紹介するのが自然であろう。

—EQL—
$\vdash \mathbf{E} = \mathbf{F}$

4.24 無デッドロック証明義務を証明する

私たちの例に戻ると、無デッドロック証明義務 DLF の仮の証明[*10]は以下のようになる。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n \leq d$
$\vdash$
$n < d \vee 0 < n$ DLF

ここで、前提 $n \leq d$ は論理和 $n < d \vee n = d$ と同値なので、推論規則 OR_L を適用する。見て分かるように[*11]、推論規則 OR_L を使うと証明が木構造になる。なぜなら、この推論規則には前件が 2 個あるからである。このような木構造は証明のありふれた形である。

$d \in \mathbb{N}$
$n \in \mathbb{N}$
$n < d \vee n = d$
$\vdash$
$n < d \vee 0 < n$
—MON—
$n < d \vee n = d$
$\vdash$
$n < d \vee 0 < n$
—OR_L—
...

...
—OR_L—
$n < d$
$\vdash$
$n < d \vee 0 < n$
—OR_R1—
$n < d | - n < d$
—HYP—

...
—OR_L—
$n = d$
$\vdash$
$n < d \vee 0 < d$
—EQ_LR—
$\vdash d < d \vee 0 < d$
—OR_R2—
$0 < d$
—?—

[*10] 訳注:「仮の証明」とは、このシークエントがそのままでは証明できず、後ほど新たな前提を追加するということです。
[*11] 訳注: 記号 ... は OR_L による分岐を表します。原文ではシークエントの列を紙面の右端で折り返すときに ... を使っています。

このシークエントは証明できないことが判明した。そのため、以下の公理を **axm0_2** という名前で追加する。

axm0_2: $0 < d$

すると、この公理の追加により、要求 FUN-2 をより詳細にする必要が生じる。

橋の上および島の上の自動車の数は制限があり、その上限は正である。[FUN-2]

デッドロック回避のために追加したこの公理はとても直観的である。なぜなら、もし d が 0 ならば、自動車が「島と橋」に入ることができないために、システムが開始と同時にデッドロックするからである。新たな公理を追加しても既存の証明はやり直さなくてもよいことを、しつこいようだが注意しておく。なぜなら、推論規則 MON (4.9 節) があるからである。

4.25 初期モデルのまとめと結論

これまで見てきたように、証明 (さらに言えば、証明の失敗) によって、イベントが単純過ぎること (4.13 節で限定条件を追加して解決した) と、定数 d についての公理が欠けていること (前節) が判明した。これはとても頻繁に起こる事例である。証明はモデルの矛盾を発見するのを助けてくれる。実際、このことはモデリング手法の心臓である！ 以下は私たちの初期モデルの最終版の状態定義である。

定数: d	**axm0_1**: $d \in \mathbb{N}$ **axm0_2**: $d > 0$	変数: n	**inv0_1**: $n \in \mathbb{N}$ **inv0_2**: $n \leq d$

そして、これが初期モデルの最終版のイベントである。

init	ML_out	ML_in
$n := 0$	**when** $n < d$ **then** $n := n + 1$ **end**	**when** $0 < n$ **then** $n := n - 1$ **end**

5　第 1 次精義化: 片側通行の橋

5.1　序論

私たちはこれから、初期モデルの***精義化*** *(refinement)* を始める。精義化は初期モデルよりも詳細なモデルを作ることである。新しいモデルは詳細であるが、初期モデルと矛盾してはならない。そのため、私たちは精義化が初期モデルと矛盾しないことを***証明*** *(prove)* しなくてはならない。この節ではその点について明らかにする。

第 1 次の精義化で、私たちは「橋」を導入する。これは私たちのシステムのより正確な観察を得ることができる。より多くのイベント、具体的には島に出入りする自動車を見ることができる。これらのイベントの名前は IL_in および IL_out とする。イベント ML_in および ML_out は引き続き存在することを注意しておく。これは、現在のモデルでは、自動車が本土を出て橋に入ること、橋を出て本土に入ることをそれぞれ意味する。これらの全ては図 5 に示されている。

5.2　状態の精義化

初期モデルでは定数 d と変数 n で作られていた状態は、これからより精密にする。定数 d はそのまま残すが、変数 n は 3 ***個の変数に置き換える*** *(replaced by three variables)*。これは、今は橋と島の上にある自動車がそれぞれ見えているが、これまでの抽象化ではそれが見えていなかったからである。さらに、私たちは自動車がどちらの方向——島または本土——に向かっているのかを見ることができる。

この理由から、状態は a, b, c の 3 個の変数から構成される。変数 a は橋の上の自動車のうち島に向かっているものの数、変数 b は島にある自動車の数、変数 c は橋の上の自動車のうち本土に向かっているものの数である。これを図 6 に示す。

初期モデルでの状態を***抽象状態*** *(abstruct state)* と呼び、精義化モデルの状態は***具象状態*** *(concrete state)* という。同様に、抽象状態の変数 n は***抽象変数*** *(abstract variable)* と呼び、具象状態の変数 a, b, c は***具象変数*** *(concrete variables)* と呼ぶ。

具象状態は***具象不変式*** *(concrete invariants)* と呼ばれるいくつかの不変式で表現される。まず、変数 a, b, c は自然数である。この条件は不変式 **inv1_1**, **inv1_2**, **inv1_3** とする。

変数: a, b, c

inv1_1: $a \in \mathbb{N}$
inv1_2: $b \in \mathbb{N}$
inv1_3: $c \in \mathbb{N}$

inv1_4: $a + b + c = n$
inv1_5: $a = 0 \lor c = 0$

続いて、これらの変数 a, b, c の和は消されてしまった変数 n に等しい。このことは不変式 **inv1_4** に示している。この不変式は変数 a, b, c と、抽象状態を表現する変数 n に関係する。最後に、私たちは橋が片側通行であること——これは私たちの基本的な要求 FUN-3 である——を言明する。これは a または c が 0 に等しいことを言えばよい。橋が片側通行であることから、明らかに a と c がどちらも 0 であってはいけないが、a と c がどちらも 0 であるときは橋が空であることを意味する。この片側通行という性質は不変式 **inv1_5** で表現される。

具象不変式のうちのいくつか (**inv1_1**, **inv1_2**, **inv1_3**, **inv1_5**) は具象変数のみを扱い、具象不変式 **inv1_4** は

抽象変数と具象変数の両方を扱うことを注意しておく。

5.3 抽象イベントの精義化

抽象イベント ML_in および ML_out は、抽象変数 n を参照せず、かわりに具象変数 a, b, c を参照するように ***精義化*** *(refined)* しなくてはならない。***具象版*** *(concrete version)* の ML_in, ML_out は以下のようになる。

ML_in
when
$0 < c$
then
$c := c - 1$
end

ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

ML_out は $a + b < d$ と $c = 0$ という 2 個の限定条件を持っていることを注意しておく。また、別の注意点として、私たちの精義化後のモデルには a, b, c の 3 個の変数があるが、それぞれのイベントで 1 個の変数しか参照していない。参照されていない変数は***変化せずに残される*** *(left unchanged)*。

これらのイベントがしていることを理解するのは簡単である。イベント ML_in の動作は、変数 c を 1 減らす、すなわち、1 台の自動車を橋から取り除く。これは本土に向かう車が橋の上に存在するとき、つまり $0 < c$ のときに可能である。なお、c が正のとき、橋の上の島に向かう自動車の数 a は 0 でなくてはならないことに注意。

イベント ML_out では、変数 a を 1 増やし、橋の上の自動車が 1 台増えるようにする。しかし、これは c が 0 のときに限って可能である。なぜなら、この橋は片側通行だからである。さらに、橋に入る自動車が「島と橋」の自動車数の上限 d を破ってはいけない。これは式 $a + b + c < d$ で表される。この式は c が 0 のとき $a + b < d$ に単純化できる。

5.4 前後述語再訪

前節で具象イベントの動作が観察されるようになったので、前後述語もより詳細にする必要がある。***複数の変数*** *(several variables)* を含むモデルでは、イベントの前後述語は動作での変更に現れない変数を明示的に考慮しなくてはならない。これはプライム付きの値がプライム無しの値に等しいという式で表される。私たちの例では、先ほど扱っていたイベントの前後述語は以下のようになる。

	ML_in	ML_out
イベント	**when** $0 < c$ **then** $c := c - 1$ **end**	**when** $a + b < d$ $c = 0$ **then** $a := a + 1$ **end**
前後述語	$a' = a \wedge b' = b \wedge c' = c$	$a' = a + 1 b' = b c' = c$

見て分かるように、動作 $c := c + 1$ の前後述語は期待通り $c' = c + 1$ を含むが、それ以外にも $a' = a$ と $b' = b$ が含まれている。動作 $a' := a$+1 についても同様である。

5.5 精義化の非公式な証明

次の節では、イベントの具象版が***抽象版の精義化である*** *(to refine its abstraction)* ためには何が必要か明確に定義する。この節ではそれを非公式に説明する。そのために、2 個のイベントの抽象版と具象版をそれぞれ比較する。以下はイベント ML_out の抽象版と具象版である。

(**抽象**_)ML_out
when
$n < d$
then
$n := n + 1$
end

(**具象**_)ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

見て分かるように、具象版の限定条件には抽象版にはまったく存在しないものがある。具象版は抽象版と***矛盾していない*** *(not contradictory)* ことは、私たちは以前から「感じて」いる。具象版のイベントが有効、すなわち限定条件の条件を満たしているとき、抽象版のイベントも確かに有効である。なぜなら、具象限定条件 $a + b < d$ および $c = 0$ は $a + b + c < d$ を導き、さらに不変式 **inv1_4** すなわち $a + b + c = n$ を用いると、抽象限定条件 $n < d$ が導かれるからである。さらに、具象版での動作 $a := a + 1$ で a が 1 増えて他の変数が変わらないとき、明らかに抽象版で n が 1 増えることと整合性がある。これは再び **inv1_4** のおかげである。同様に、イベント ML_in の抽象版と具象版を以下に示す。

(**抽象**_)ML_in
when
$0 < n$
then
$n := n - 1$
end

(**具象**_)ML_in
when
$0 < c$
then
$c := c - 1$
end

抽象版と具象版の ML_out についての非公式な「証明」は、具象版が抽象版と矛盾しないことをから得られる。

5.6 抽象イベントの精義化の正しさを証明する

この節では具象イベントが本当に抽象イベントの精義化であることを保証するために証明すべきものは何かを正確に定義する体系的な規則を紹介する。ここでは異なる二つのものを証明することになる。一つは限定条件についての言明、もう一つは動作についての言明である。

限定条件の強化

私たちはまず具象限定条件が抽象限定条件よりも***強い*** *(stronger)* ことを証明する。「強い」という用語は具象限定条件が抽象限定条件を***含む*** *(impliy)* ことを意味する。別の言い方をすると、抽象限定条件が有効でないとき具象限定条件が有効になってはいけないということである。一方で、具象遷移に対応する遷移が抽象モデ

ルに無くてもよい。限定条件の強化は、定数の抽象公理、 抽象不変式および具象不変式から証明される。なお、第 5.16 節で見るように、限定条件の精錬化を強くし過ぎてはいけない。なぜなら、それによってデッドロックが起こることがあるからである。

さらに普遍化すると以下のようになる。c を定数、$A(c)$ を公理の集合、v を抽象変数、$I(c,v)$ を抽象不変式の集合、w を具象変数、$J(c,v,w)$ を具象不変式の集合とする。抽象イベントの限定条件の集合を $G(c,v)$ とする。言い換えれば、$G(c,v)$ は列 $G_1(c,v), G_2(c,v)\ldots$ を表す。具象イベントの限定条件の集合を $H(c,w)$ とする。私たちは以下のシークエントを具象限定条件 $G_i(c,v)$ のそれぞれについて証明する必要がある。

$A(c)$	公理
$I(c,v)$	抽象不変式
$J(c,v,w)$	具象不変式
$H(c,w)$	具象限定条件
$\vdash$	$\vdash$
$G_i(c,v)$ [GRD]	抽象限定条件 [GRD]

重ねて注意するが、具象不変式の集合 $J(c,v,w)$ は具象変数 w のみを含む初等的な不変式もあるが、抽象変数 v と具象変数 w をどちらも含む不変式もある。これが具象不変式の集合を $J(c,v,w)$ と書く理由である。

さらに注意しておくと、精義化で新しい定数を追加することもできる。しかしながら、式を複雑にしないために、具象不定式 $J(c,v,w)$ の言明には含めないことにする。

正しい精義化

具象イベントが具象変数 w を w' に変更する際に抽象イベントと矛盾しないことを証明する必要がある。この遷移が起こるとき、抽象イベントは抽象変数 v[*12]を v' に変更する。v は具象不変式 $J(c,v,w)$ で w に関連付けられており、v' は変更後の具象不変式 $J(c,v',w')$ で w' に関連付けられている。これは以下のダイヤグラム (訳文では省略) に書くことができる。

私たちの決まりでは、INV という名前の証明義務規則を以下のように書くことができる。ただし $J_i(c,v,w)$ は具象不変式の集合 $J(c,v,w)$ のうちの 1 個の不変式である。

$A(c)$	公理
$I(c,v)$	抽象不変式
$J(c,v,w)$	具象不変式
$H(c,w)$	具象限定条件
$\vdash$	$\vdash$
$J_i(c,E(c,v),F(c,w))$ [INV]	変更後の具象不変式 [INV]

5.7 精義化規則を適用する

私たちの例に戻って、精義化後のイベント ML_in と ML_out それぞれに規則 GRD を適用する。ここで得られるシークエントは複雑に見えるが、証明は簡単である。

[*12] 訳注: 原文では concrete variables v と言っているが、間違いと思われます。

イベント ML_out **の限定条件強化**

以下は ML_out に関わる証明義務である。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq d$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	ML_out / GRD
inv1_4	$a + b + c = n$	
inv1_5	$a = 0 \lor c = 0$	
ML_out の抽象限定条件	$a + b < d$	
⊢	$c = 0$	
ML_out の具象限定条件	⊢	
	$n < d$	

この巨大な印象のあるシークエントは劇的に単純化できる。まず、使わない仮説を MON で除去する。次に、仮説の等式 $c = 0$ を使って、仮説 $a + b + c = n$ を $a + b + 0 = n$ さらに $a + b = n$ に変形する。さらに、この等式を仮説 $a + b < d$ に適用して $n < d$ を得る。これは証明したい結論とまったく同じなので、HYP を適用する。形式的には、シークエントの変形は以下のように書ける (ただし MON を適用したところから)

$a + b + c = n$
$a + b < d$
$c = 0$
⊢
$n < d$
—EQ_LR—
$a + b + 0 = n$ [ARI]
$a + b < d$
⊢
$n < d$
—ARI—
$a + b = n$
$a + b < d$
⊢
$n < d$
—EQ_LR—
$n < d$
⊢
$n < d$
—HYP—

見て分かるとおり、私たちは総称的な推論規則 ARI を導入する。これは、証明が単純な算術的性質によることを形式主義的にならずに言明するものである。個別の推論規則を定義してもよいが、ここではあまり重要で

はない。どの算術的性質を用いたか明確にするために、関係する仮説または結論を下線[*13]で示す。

ML_in の限定条件強化

MON を適用した後のシークエントを以下に示す。

$$
\begin{array}{l}
b \in \mathbb{N} \\
a + b + c = n \\
a = 0 \lor c = 0 \\
0 < c \\
\vdash \\
0 < n
\end{array}
$$

論理和の仮説は場合分けによる証明を示唆している。その後は等式を適用することでシークエントを単純化できる。最後にいくつかの算術的な変形を行う。このシークエントの証明を以下に示す。

$$
\begin{array}{l}
b \in \mathbb{N} \\
a + b + c = n \\
a = 0 \lor c = 0 \\
0 < c \\
\vdash \\
0 < n \\
\text{—OR\_L—} \\
\cdots
\end{array}
$$

[*13] 訳注: 訳文では述語の右側に [ARI] マークを付けています。

$\cdots$
—OR_L—
$b \in \mathbb{N}$
$a + b + c = n$
$a = 0$
$0 < c$
$\vdash$
$0 < n$
—EQ_LR—
$b \in \mathbb{N}$
$0 + b + c = n$ [ARI]
$0 < c$
$\vdash$
$0 < n$
—ARI—
$b \in \mathbb{N}$ [ARI]
$b + c < n$ [ARI]
$0 < c$
$\vdash$
$0 < n$
—ARI—
$c \leq n$ [ARI]
$0 < c$ [ARI]
$\vdash$
$0 < n$
—ARI—
$0 < n$
$\vdash$
$0 < n$
—HYP—

$\cdots$
—OR_L—
$b \in \mathbb{N}$
$a + b + c = n$
$c = 0$
$0 < c$
$\vdash$
$0 < n$
—EQ_LR—
$b \in \mathbb{N}$
$a + b + 0 = \mathbb{N}$
$0 < 0$
$\vdash$
$0 < n$
—MON—
$0 < 0$ [ARI]
$\vdash$
$0 < n$
—ARI—
$\bot$
$\vdash$
$0 < n$
—FALSE_L—

証明義務規則 INV は 2 個のイベントと 5 個の具象不変式から 10 個の述語を生み出すが、そのうちの一部を示し、残りは練習問題とする。

イベント ML_out **での不変式 inv1_4 の保存**

MON 適用後からの証明を以下に示す。

$a + b + c = n$
$\vdash$
$a + 1 + b + c = n + 1$ [ARI]
—ARI—
$a + b + c = n$
$\vdash$
$a + b + c + 1 = n + 1$
—EQ_LR—
$\vdash n + 1 = n + 1$
—EQL—

イベント ML_in **での不変式 inv1_5 の保存**

MON 適用後からの証明を以下に示す。

$a = 0 \lor c = 0$
$0 < c$
$\vdash$
$a = 0 \lor c - 1 = 0$
—OR_L—
...

...
—OR_L—
$a = 0$
$0 < c$
$\vdash$
$a = 0 \lor c - 1 = 0$
—OR_R1—
$a = 0$
$0 < c$
$\vdash$
$a = 0$
—MON— $a = 0$
$\vdash$
$a = 0$
—HYP—

...
—OR_L—
$c = 0$
$0 < c$
$\vdash$
$a = 0 \lor c - 1 = 0$
—EQ_LR—
$0 < 0$ [ARI]
$\vdash$
$a = 0 \lor c - 1 = 0$
—ARI—
$\bot \vdash a = 0 \lor c - 1 = 0$
—FALSE_L—

5.8 初期化イベント init の精義化

私たちは特殊イベント init の精義化も定義しなくてはならない。このイベントは明らかに以下のようになる。

init
  $a := 0$
  $b := 0$
  $c := 0$

私たちは***多重動作*** *(multiple action)* を初めて見た。前後述語は以下のようになる。

$a' = 0 \land b' = 0 \land c' = 0$

5.9 初期化イベント init のための証明義務の精義化規則

初期化イベント init に適用する証明義務規則は INV の特殊な場合である。抽象初期化が後述語 $v' = K(c)$ を持ち、具象初期化が後述語 $w' = L(c)$ を持つとき、証明義務規則は以下のようになる。

$A(c)$	公理	
⊢	⊢	
$J_i(c, K(c), L(c))$	変更後の具象不変式	

限定条件強化についての証明義務規則を用いていないことに注意。初期化イベントには定義より限定条件が存在しない。

5.10 初期化のための証明義務精義化規則を適用する

前節で導入した証明義務規則の適用は、私たちの例では一本道である。5 個の述語のうち重要なものだけを示す。まずは不変式 **inv1_4** について。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$d > 0$	**inv1_4** / INV
⊢	⊢	
変更後の具象不変式 **inv1_4**	$0 + 0 + 0 = 0$	

以下は不変式 **inv1_5** についての証明義務である。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$d > 0$	**inv1_5** / INV
⊢	⊢	
変更後の具象不変式 **inv1_5**	$0 = 0 \lor 0 = 0$	

証明は読者に任せる。

5.11 新しいイベントの導入

ここで私たちは自動車が島に入ったり島から出たりするイベントを導入する。

IN_in
when
$0 < a$
then
$a := a - 1$
$b := b + 1$
end

IL_out
when
$0 < b$
$a = 0$
then
$b := b - 1$
$c := c + 1$
end

ここで再び ***多重動作*** *(multiple actions)* が登場する。ただし、この場合にも動作は網羅的ではなく、イベント ML_in では変数 c が、イベント ML_out では変数 a が欠けている。このような多重動作は以下のような前後述語に関連付けられる。

$a' = a - 1 \land b' = b + 1 \land c' = c$ $\qquad$ $a' = a \land b' = b - 1 \land c' = c + 1$

これらのイベントが何をしているか理解するのは簡単である。イベント IL_in は自動車が橋を離れて島に入ることを表す。そのため、このイベントの動作は橋の自動車の数 a を減らし、島の自動車の数 b を減らす。これは橋に自動車があるとき、すなわち $0 < a$ のときのみ可能である。

イベント IL_out は自動車が島を離れて橋に入ることを表す。このイベントの動作は明らかに b を減らし c を増やす。これは島に自動車がある、すなわち $0 < b$ のときのみ可能である。イベント IL_out の 2 番目の条件は橋の上に島に向かう自動車がないことで、——私たちの橋は片側通行である——これは条件式 $a = 0$ に相当する。

5.12 空の動作 skip

次の節で説明するように、新しく導入されたイベントは「ダミーイベント」の精錬化であることを証明することになる。ダミーイベントは限定条件を持たず、***何もしない*** *(does nothing)*。このような何もしない動作は空の動作 skip と表記する。

これは非常に重要な注意点なのだが、空の動作 skip の前後述語はモデルの状態に依存する。私たちの例では、初期モデルの変数は n のみであるので、前後述語は以下のようになる。

$$n' = n$$

しかし、具象変数についての空の動作 skip は、具象変数が a, b, c の 3 個なので、以下のような別の前後述語になる。

$$a' = a \wedge b' = b \wedge c' = c$$

5.13 新しいイベントの正しさを証明する

新しく導入したイベント (5.11 節) は抽象モデルでは見えない。しかし、それは存在しているし発生している。顕微鏡を使わなければ見えないものが顕微鏡を使えば見えるのと同じである。この比喩では、精錬化はシステムを顕微鏡を通してみることである。新しいイベント ML_in および ML_out による遷移は抽象モデルでは見えないが、存在している。この考えを定式化すると、新しいイベントは限定条件を持たず空の動作 skip を持つイベントの精錬化であるということになる。結論として、私たちは証明義務規則 INV を新しいイベントの正しさを証明するために使うことができる。以下はイベント IL_in と具象不変式 inv1_4 についての証明義務である。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq d$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	IL_in / **inv1_4** / INV
inv1_3	$c \in \mathbb{N}$	
inv1_4	$a + b + c = n$	
inv1_5	$a = 0 \lor c = 0$	
イベント ML_in の具象限定条件	$0 < a$	
⊢	⊢	
変更後の不変式 **inv1_4**	$a - 1 + b + 1 + c = n$	

変数 n は限定条件を持たず空の動作 skip を持つ抽象イベントの精義化では変更されないことを注意しておく。導出規則 MON の適用後は以下のような証明を得る。

$a + b + c = n$
⊢
$a - 1 + b + 1 + c = n$ [ARI]
—ARI—
$a + b + c = n$
⊢
$a + b + c = n$
—HYP—

以下はイベント IL_in と具象不変式 **inv1_5** についての証明義務である。

...	...	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	
inv1_4	$a + b + c = n$	IL_in / **inv1_5** / INV
inv1_5	$a = 0 \lor c = 0$	
IL_in の具象限定条件	$0 < a$	
⊢	⊢	
変更後の不変式 **inv1_5**	$a - 1 = 0 \lor c = 0$	

MON の適用後からの証明は以下。

$a = 0 \lor c = 0$
$0 < a$
⊢
$a - 1 = 0 \lor c = 0$
—OR_L—
...

```
...
—OR_L—
a = 0
0 < a
⊢
a − 1 = 0 ∨ c = 0
—EQ_LR—
0 < 0 [ARI]
⊢
−1 = 0 ∨ c = 0
—ARI—
⊥
⊢
−1 = 0 ∨ c = 0
—FALSE_L—
```

```
...
—OR_L—
c = 0
0 < a
⊢
a − 1 = 0 ∨ c = 0
—OR_R2—
c = 0
0 < a
⊢
c = 0
—MON—
c = 0
⊢
c = 0
—HYP—
```

これ以外の述語やイベント IL_out についての証明義務は読者の練習問題とする。

5.14 新しいイベントの収束を証明する

新しいイベントを導入する場合には「発散しない」(do not diverge) ことを証明する必要がある。言い換えれば、新しいイベントは限りなく (indefinitely) 可能 (enabled) であってはならない。もしそれを許せば、既存の (existing) イベントの具象版——ML_out と ML_in——は限りなく延期されてしまう。既存のイベントは抽象モデルでは発生し得るのだから、限りない延期は確かに避けなければならない。これを証明するため、私たちは ***変量*** *(variant)* と呼ばれる自然数の式を明示し、それが ***全ての新しいイベントで減少する*** *(decreased by all new events)* ことを証明する。これは 2 個の証明義務規則をもたらす。第 1 に、提示された変量が自然数であること、第 2 に、変量が全ての新しいイベントで減少することである。

第 1 の証明義務規則——名前は NAT——は、明示された変量 $V(c,w)$ が自然数であることである。この証明は、定数 c の公理 $A(c)$、抽象不変式 $I(c,v)$、具象不変式 $J(c,v,w)$、および、新しいイベントそれぞれの限定条件 $H(c,w)$ を前提とする。限定条件 $H(c,w)$ を前件 (シークエントの前提) に置くのは、限定条件の条件式が真でないときは、変量が自然数であることの証明に興味がないからである。

```
A(c)                    公理
I(c, v)                 抽象不変式
J(c, v, w)              具象不変式
H(c, w)                 新しいイベントの具象限定条件
⊢                       ⊢
V(c, w) ∈ ℕ [NAT]       変量が自然数 [NAT]
```

第 2 の証明義務規則は変量 $V(c,w)$ が減少することである。これは、新しいイベントのそれぞれに対して、限定条件 $H(c,w)$ と前後述語 $w' = F(c,w)$ を前提として証明する。

$A(c)$	公理
$I(c,v)$	抽象不変式
$J(c,v,w)$	具象不変式
$H(c,w)$	新しいイベントの具象限定条件
⊢	⊢
$V(c,F(c,w)) < V(c,w)$ [VAR]	変更後の変量 < 変量 [VAR]

変量が 1 個しかないことを注意しておく。言い換えれば、それぞれの新しいイベントは***同じ変量*** *(same variant)* を減少させなければならない。変量は自然数の式よりも少し複雑になることもあるが、この例ではこれで十分である。

5.15 収束証明義務を適用する

私たちの例では、提案する変量は以下のようになる。

variant_1: $2 * a + b$

証明義務規則 VAR をイベント IL_in に適用すると、長いが易しい以下のような証明になる。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq f$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	IL_in / VAR
inv1_4	$a + b + c = \mathbb{N}$	
inv1_5	$a = 0 \vee c = 0$	
イベント **IL_in** の具象限定条件	$0 < a$	
⊢	⊢	
変更後の変量 < 変量	$2 * (a - 1) + b + 1 < 2 * a + b$	

証明義務規則 VAR をイベント IL_out に適用すると以下。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq f$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	IL_out / VAR
inv1_4	$a + b + c = \mathbb{N}$	
inv1_5	$a = 0 \vee c = 0$	
イベント IL_out の具象限定条件	$0 < b$ $a = 0$	
⊢	⊢	
変更後の変量 < 変量	$2 * a + b - 1 < 2 * a + b$	

これらのシークエントは単純な算術計算だけで証明できる。具体的な証明はここでは省略する。

5.16 相対的無デッドロック

最後に、私たちは全ての具象イベント (新しいイベントも含めて) が抽象イベントと比較してデッドロックしないことを証明する必要がある。そのためには、抽象限定条件 $G_1(c,v),\ldots,G_m(c,v)$ の論理和が具象限定条件 $H_1(c,w),\ldots,H_n(c,w)$ の論理和を導くことを証明する。この証明義務規則は DLF という名前で、以下のようになる。

$A(c)$	公理
$I(c,w)$	抽象不変式
$J(c,v,w)$	具象不変式
$G_1(c,v) \lor \ldots \lor G_m(c,v)$	抽象限定条件の論理和
$\vdash$	$\vdash$
$H_1(c,w) \lor ... \lor H_n(c,w)$ [DLF]	具象限定条件の論理和 [DLF]

5.17 相対的無デッドロック証明義務規則を適用する

証明義務規則 DLF を適用すると以下のような証明になる。抽象限定条件の論理和を前件から除去していることを注意しておく。抽象限定条件の論理和は初期モデルの段階で証明されているからである。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq d$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	DLF
inv1_4	$a + b + c = n$	
inv1_5	$a = 0 \lor c = 0$	
$\vdash$	$\vdash$	
具象限定条件の論理和	$(a + b < dc = 0) \lor c > 0 \lor a > 0 \lor (b > 0a = 0)$	

5.18 新しい推論規則

前節のシークエントでは、さまざまな述語の***論理和*** *(disjunction)* を証明する必要がある。これをするために便利な方法は、結論のいずれかの項の否定を前提に移すことである。(論理和の交換規則と結合規則は使用できるものとする。) この推論規則を以下に示す[*14]。

$\mathbf{H}, \neg\mathbf{P} \vdash \mathbf{Q}$
—OR_R—
$\mathbf{H} \vdash \mathbf{P} \lor \mathbf{Q}$

[*14] 訳注: 推論規則は下から上に進むように書かれています。

以下の 2 個の推論規則は、シークエントの前提 (AND_L) および結論 (AND_R) に現れる論理積形の述語を単純化するものである。

$$\frac{\mathbf{H}, \mathbf{P}, \mathbf{Q} \vdash \mathbf{R}}{\mathbf{H}, \mathbf{P} \wedge \mathbf{Q} \vdash \mathbf{R}}\ \text{AND\_L}$$

$$\frac{\mathbf{H} \vdash \mathbf{P} \quad \mathbf{H} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \wedge \mathbf{Q}}\ \text{AND\_R}$$

無デッドロック証明義務を証明する

新しい推論規則を得たことで、無デッドロック証明義務は以下のように証明できる。(MON 適用後から示す)

$$\frac{\begin{array}{l} 0 < d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \end{array} \vdash (a + b < d \wedge c = 0) \vee c > 0 \vee a > 0 \vee (b > 0 \wedge a = 0)}{\text{OR\_R}}$$

$$\frac{\begin{array}{l} 0 < d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N}\ [\text{ARI}] \\ \neg(c > 0)\ [\text{ARI}] \end{array} \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{\text{ARI}}$$

$$\frac{\begin{array}{l} 0 < d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \\ c = 0 \end{array} \vdash (a + b < d \wedge c = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{\text{EQ\_LR}}$$

$$\frac{\begin{array}{l} 0 < d \\ a \in \mathbb{N} \\ b \in \mathbb{N} \end{array} \vdash (a + b < d \wedge 0 = 0) \vee a > 0 \vee (b > 0 \wedge a = 0)}{\text{OR\_R}}$$

$0 < d$
$a \in \mathbb{N}$ [ARI]
$b \in \mathbb{N}$
$\neg(a > 0)$ [ARI]
$\vdash$
$(a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)$
—ARI—
$0 < d$
$b \in \mathbb{N}$
$a = 0$
$\vdash$
$(a + b < d \wedge 0 = 0) \vee (b > 0 \wedge a = 0)$
—EQ_LR—
$0 < d$
$b \in \mathbb{N}$ [ARI]
$\vdash$
$(0 + b$ [ARI] $< d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)$
—ARI—
$0 < d$
$b = 0 \vee b > 0$
$\vdash$
$(b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)$
—OR_L—
...

...
—OR_L—
$0 < d$
$b = 0$
$\vdash$
$(b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)$
—OR_R1—
$0 < d$
$b = 0$
$\vdash$
$b < d \wedge 0 = 0$
—EQ_LR—
$0 < d$
$\vdash$
$0 < d \wedge 0 = 0$
—AND_R—
...

...	...
—AND_R—	—AND_R—
$0 < d \vdash 0 < d$	$0 < d \vdash 0 = 0$
—HYP—	—EQL—

...
—OR_L—
$0 < d$
$b > 0$
$\vdash$
$(b < d \wedge 0 = 0) \vee (b > 0 \wedge 0 = 0)$
—OR_R2—
$0 < d, b > 0 \vdash b > 0 \wedge 0 = 0$
—AND_R—
...

...	...
—AND_R—	—AND_R—
$0 < d, b > 0 \vdash b > 0$	$0 < d, b > 0 \vdash 0 = 0$
—HYP—	—EQL—

5.19 第 1 次精義化のまとめ

以下は第 1 次精義化の状態のまとめである。

定数: d
変数: a, b, c

inv1_1: $a \in \mathbb{N}$
inv1_2: $b \in \mathbb{N}$
inv1_3: $c \in \mathbb{N}$
inv1_4: $a + b + c = \mathbb{N}$
inv1_5: $a = 0 \lor c = 0$

variant1: $2 * a + b$

以下は第 1 次精義化のイベントのまとめである。

ML_in
when
$0 < c$
then
$c := c - 1$
end

ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

IL_in
when
$0 < a$
then
$a := a - 1$
$b := b + 1$
end

IL_out
when
$0 < b$
$a = 0$ **then**
$b := b - 1$
$c := c - 1$
end

init
$a := 0$
$b := 0$
$c := 0$

6　第 2 次精細化: 信号機の導入

現状では、橋のモデルはちょっと不可解に見える。私たちの観察では、自動車の運転手たちは自動車の数を数えて、本土 (イベント ML_out) または島 (イベント IL_out) から橋に移るかどうかを決めているように見える。このことは、運転手たちがシステムの状態を観測できることを意味している。明らかに、これはリアルではない。現実には、当たり前だが、運転手たちは信号機の指示に従っているのであり、自動車を数えているわけではない！

この精細化では、まず 2 個の信号機を導入し、ml_tl と il_tl と名付け、不変式を作り直し、最後に信号機の色を変えるイベントを導入する。図 7 は、この精細化で観測できるようになる物理的な状況を示している。

6.1　状態の精細化

この段階では、私たちは定数の集合を、最初に紹介する集合 *COLOR* と 2 個の異なる値「赤」(*red*) および「青」 (*green*) で拡張しなくてはならない。結果は以下のようになる。

集合: *COLOR*
定数: *red*, *green*

axm2_1: $COLOR = \{green, red\}$
axm2_2: $green \neq red$

その上で、2 個の新しい変数 ml_tl (本土の信号機) と il_tl (島の信号機) を導入する。これらの変数は色で定義される。このことは不変式 **inv2_1** と **inv2_2** で定式化する。信号機が青のときのみ運転手が通過できるように、2 個の***条件付き不変式*** *(conditional invariants)* **inv2_3** と **inv2_4** を導入する。これは、*ml_tl* が青のときイベント ML_out の抽象限定条件が真となり、*il_tl* が青のときイベント Il_out の抽象限定条件が真となることを保証する。これは要求 ENV_1, ENV_2, ENV_3 を表現することを注意しておく。

変数: ...
　ml_tl
　il_tl

inv2_1: $ml\_tl \in COLOR$
inv2_2: $il\_tl \in COLOR$
inv2_3: $ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
inv2_4: $il\_tl = green \Rightarrow 0 < ba = 0$

不変式 **inv2_3** と **inv2_4** は***条件付き不変式*** *(conditional invariants)* であることを再び注意しておく。これらの不変式は含意演算子 $\Rightarrow$ を使って導入する。明らかに、この論理演算に対応する推論規則が必要である。それは 6.6 節で説明する。

変数について見ると、今回の精細化は前回とは少し異なる。前回は、具象変数 a, b, c が抽象変数 n を置き換えた。今回は、変数 *ml_tl* と *il_tl* を追加し、抽象変数 a, b, c もそのまま利用する。このような精細化手法を***重ね合わせ*** *(superposition)* という。重ね合わせには追加の証明義務規則が必要であることを 6.4 節で説明する。

6.2　イベントの精細化

イベント ML_out と IL_out は、限定条件が信号機の色を参照するように変更する。私たちは自動車の運転手が信号機に従うことを暗黙の前提とする。これは要求 ENV-3 に示されていることである。イベント IL_in および ML_in は今回の精細化で変更されないことを注意しておく。以下は抽象版と具象版の ML_out である。

(**抽象**_)ML_out
when
$c = 0$
$a + b < d$
then
$a := 0$
end

(**具象**_)ML_out
when
$ml\_tl = green$
then
$a := 0$
end

以下はイベント IL_out の抽象版と具象版である。

(**抽象**_)IL_out
when
$a := 0$
$0 < b$
then
$b, c := b - 1, c + 1$
end

(**具象**_)IL_out
when
$il\_tl = green$
then
$b, c := b - 1, c + 1$
end

6.3 新しいイベントの導入

私たちは 2 個の新しいイベントを導入する。それらのイベントは、信号機の色が赤で、条件を満たすとき信号機を青に変える。その条件は抽象イベント ML_out および IL_out の抽象限定条件と全く同じである。新しいイベントを以下に示す。

ML_tl_green
when
$ml\_tl = red$
$a + b < d$
$c = 0$
then
$ml\_tl = green$
end

IL_tl_green
when
$il\_tl = red$
$0 < b$
$a = 0$
then
$il\_tl = green$
end

6.4 重ね合わせ: 精義化規則を適合させる

この節では、私たちの例から離れ、重ね合わせについて普遍化して説明する。重ね合わせ、すなわち抽象変数が具象モデルの状態に残っている場合、証明義務規則 INV をそれに適合させる必要がある。他の精義化規則はそのまま使用できる。

抽象状態には変数 u と v があり、具象状態には変数 v と w があるとする。すなわち、変数 v は抽象状態と具象状態に共通である。$I(c, u, v)$ を抽象不変式、$J(c, u, v, w)$ を具象不変式とする。証明義務規則 INV を適用するには、抽象状態と具象状態は***完全に共通部分を持たない*** *(completely disjoint)* 必要があり、それは重ね合わせの場合には当てはまらない。

共通部分を持たない世界に戻るために、具象状態の変数を改名する。例えば、変数 v を v_1 に名前を変えて、**追加の具象不等式 $v_1 = v$ を加える** *(adding the additional concrete invariant)*。抽象状態でのイベントの前後述語が $u' = E(c,u,v)$ と $v' = M(c,u,v)$ とする。対応する具象イベントの具象状態での前後述語が $v' = N(c,v,w)$ と $w' = F(c,v,w)$ とする。この具象イベントの限定条件は $H(c,v,w)$ とする。証明義務規則 INV を適用すると、以下の 2 通りのシークエントを得る。

定数についての公理	$A(c)$	$A(c)$
抽象不変式	$I(c,u,v)$	$I(c,u,v)$
具象不変式	$J(c,u,v1,w)$	$J(c,u,v_1,w)$
具象限定条件	$v_1 = v$	$v_1 = v$
$\vdash$	$H(c,v_1,w)$	$H(c,v_1,w)$
変更後の不変式	$\vdash$	$\vdash$
	$J_i(c,E(c,u,v),M(c,u,v),F(c,v_1,w))$	$M(c,u,v) = N(c,v_1,w)$

ここで等式 $v_1 = v$ を適用すると、前掲のシークエントに出現する全ての v_1 が v に置換され、以下のシークエントでは v_1 は出現しなくなる。これが証明義務規則 INV の私たちが望んだ適用結果である。この結果は、既出の証明義務規則 INV に加えて別の証明義務規則を導入していると解釈できる。後者は、抽象状態と具象状態が共有している変数に関連付けられている抽象および具象の式[*15]は、具象不変式 $J(c,u,v,w)$ の前提のもとで等しいというものである。

$A(c)$	$A(c)$
$I(c,u,v)$	$I(c,u,v)$
$J(c,u,v,w)$	$J(c,u,v,w)$
$H(c,v,w)$	$H(c,v,w)$
$\vdash$	$\vdash$
$J(c,E(c,u,v),M(c,u,v),F(c,v,w))$	$M(c,u,v) = N(c,v,w)$

前者のシークエントは引き続き証明義務規則 INV と呼び、後者は新しく SIM と呼ぶ。この名前の由来は、共通の変数に関連付けられている抽象と具象の式が等しいことである。

$A(c)$	定数についての公理
$I(c,u,v)$	抽象不変式
$J(c,u,v,w)$	具象不変式
$H(c,v,w)$	具象限定条件
$\vdash$	$\vdash$
$M(c,u,v) = N(c,v,w)$ [SIM]	共通の変数に関連付けられている式が等しい [SIM]

6.5 イベントの正しさの証明

古い[*16]具象イベントが抽象イベントの正しい精錬化であることを証明するには、証明義務規則 GRD, SIM, INV を適用すればよい。

[*15] 訳注: 前後述語の右辺。
[*16] 訳注: 対応する抽象イベントが存在する。

イベント IL_in と ML_in は抽象版と同じである。そのため、証明義務 GRD と SIM は証明すべきものを何も生成しない。証明義務規則 INV が導く言明も他愛ない。具象不変式 **inv2_3** と **inv2_4** が保存されることを示せばよい。

証明義務 SIM をイベント IL_out と ML_out に適用した結果も他愛ない。なぜなら抽象版と具象版で動作は全く同じだからである。証明義務規則 GRD についても証明は簡単である。不変式 **inv2_3** と **inv2_4** から限定条件強化を証明できる。これはすでに 6.2 節で非公式に示していることである。

すると、いま残っている証明義務は、イベント IL_out と ML_out に証明義務規則 INV を適用したものである。これから見ていくように、この証明はある厄介事を引き起こす。

6.6 新たな論理推論規則

この節では、証明に必要な新しい論理推論規則をいくつか導入する。以下の 2 個の規則は、前提および結論の、論理包含を含む述語を単純化する。加えて、否定を含む前提を扱う規則を示す[*17]。

$$\frac{\mathbf{H}, \mathbf{P}, \mathbf{Q} \vdash \mathbf{R}}{\mathbf{H}, \mathbf{P}, \mathbf{P} \Rightarrow \mathbf{Q} \vdash \mathbf{R}}\ \text{IMP\_L}$$

$$\frac{\mathbf{H}, \mathbf{P} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \Rightarrow \mathbf{Q}}\ \text{IMP\_R}$$

$$\frac{\mathbf{H}, \neg\mathbf{Q} \vdash \mathbf{P}}{\mathbf{H}, \neg\mathbf{P} \vdash \mathbf{Q}}\ \text{NOT\_L}$$

6.7 仮の証明と解決

イベント ML_out **が不変式 inv2_4 を保存することを証明する。**

証明すべきシークエントは以下。

[*17] 全ての推論規則は第 9 章にまとめてある。

axm0_1	$d \in \mathbb{N}$	
axm0_2	$0 < d$	
axm2_1	$COLOR = \{green, red\}$	
axm2_2	$green \neq red$	
inv0_1	$n \in \mathbb{N}$	
inv0_2	$n \leq d$	
inv1_1	$a \in \mathbb{N}$	
inv1_2	$b \in \mathbb{N}$	
inv1_3	$c \in \mathbb{N}$	
inv1_4	$a + b + c = \mathbb{N}$	ML_out / **inv2_4** / INV
inv1_5	$a = 0 \vee c = 0$	
inv2_1	$ml\_tl \in \{red, green\}$	
inv2_2	$il\_tl \in \{red, green\}$	
inv2_2	$ml\_tl = green \Rightarrow a + b < d \wedge c =$	
inv2_3	0	
inv2_4	$il\_tl = green \Rightarrow 0 < b \wedge a = 0$	
ML_out の限定条件	$ml\_tl = green$	
⊢	⊢	
変更後の ML_out	$il\_tl = green \Rightarrow 0 < b \wedge a + 1 = 0$	

以下は仮の証明である (MON 適用後からの)

$green \neq red$
$il\_tl = green \Rightarrow 0 < b \wedge a = 0$
$ml\_tl = green$
⊢
$il\_tl = green \Rightarrow 0 < b \wedge a + 1 = 0$
—IMP_R—
$green \neq red$
$il\_tl = green \Rightarrow 0 < b \wedge a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$0 < b \wedge a + 1 = 0$
—IMP_L—
$green \neq red$
$0 < b \wedge a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$0 < b \wedge a + 1 = 0$
—AND_L—

$green \neq red$
$0 < b$
$a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$0 < b \wedge red$
—AND_R—
...

...
—AND_R—
$green \neq red$
$0 < b$
$a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$0 < b$
—MON—
$0 < b$
⊢
$0 < b$
—HYP—

...
—AND_R—
$green \neq red$
$0 < b$
$a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$a + 1 = 0$
—EQ_LR—
$green \neq red$
$0 < b$
$a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$0 + 1 = 0$ [ARI]
—ARI—
$green \neq red$
$0 < b$
$a = 0$
$ml\_tl = green$
$il\_tl = green$
⊢
$1 = 0$
—?—

最後のシークエントは証明不可能である。

イベント IL_out **が不変式 inv2_3 を保存することを証明する。**

イベント IL_out が不変式 **inv2_3** を保存することの証明は以下。

...	...	
axm2_1	$COLOR = \{red, green\}$	
axm2_2	$green \neq red$	
inv2_1	$ml\_tl \in \{red, green\}$	
inv2_2	$il\_tl \in \{red, green\}$	
inv2_3	$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$	IL_out / **inv2_3** / INV
inv2_4	$il\_tl = green \Rightarrow 0 < b \wedge a = 0$	
IL_out の限定条件	$il\_tl = green$	
⊢	⊢	
変更後の **inv2_3**	$ml\_tl = green \Rightarrow a + b - 1 < d \wedge c + 1 = 0$	

仮の証明は以下 (MON 適用後)。

$green \neq red$
$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
$il\_tl = green$
⊢
$ml\_tl = green \Rightarrow a + b - 1 < d \wedge c + 1 = 0$
—IMP_R—
$green \neq red$
$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
$il\_tl = green$
$ml\_tl = green$
⊢
$a + b - 1 < d \wedge c + 1 = 0$
—IMP_L—
$green \neq red$
$a + b < d \wedge c = 0$
$il\_tl = green$
$ml\_tl = green$
⊢
$a + b - 1 < d \wedge c + 1 = 0$
—AND_L—
$green \neq red$
$a + b < d$
$c = 0$
$il\_tl = green$
$ml\_tl = green$
⊢
$a + b - 1 < d \wedge c + 1 = 0$
—AND_R—
...

...
—AND_R—
$green \neq red$
$a + b < d$
$c = 0$
$il\_tl = green$
$ml\_tl = green$
⊢
$a + b - 1 < d$
—MON—
$a + b < d | - a + b - 1 < d$
—DEC—

...
—AND_R—
$green \neq red$
$c = 0$
$il\_tl = green$
$ml\_tl = green$
⊢
$c + 1 = 0$
—EQ_LR—
$green \neq red$
$il\_tl = green$
$ml\_tl = green$
⊢
$0 + 1 = 0$ [ARI]
—ARI—
$green \neq red$
$il\_tl = green$
$ml\_tl = green$
⊢
$1 = 0$
—?—

明らかに最後のシークエントは証明不可能である。

解決

上記の 2 個の証明は以下のシークエントが原因で失敗する。

$green \neq red$
$il\_tl = green$
$ml\_tl = green$
⊢
$1 = 0$

これが示すのは、両方の信号機が同時に青になってはいけないということである。これは***明白な事実*** *(obvious fact)* であるが、私たちは明記するのを忘れていた。ここで以下の不変式を追加する。

inv2_5: $ml\_tl = red \vee il\_tl = red$

この不変式を要求に加えてもよい。しかし、要求 ENV-3「自動車は赤信号を通過せず、青信号だけを通ることを前提とする」と要求 FUN-3「橋は片側通行であり、***同時に両方向*** *(not both at the same time)* に通行することはできない」から演繹することもできる。この不変式を追加することで、私たちの証明は以下のように拡張され、問題は解決する。

green ≠ *red*
ml_tl = *red* ∨ *il_tl* = *red*
il_tl = *green*
ml_tl = *green*
⊢
1 = 0
—OR_L—
...

...
—OR_L—
green ≠ *red*
ml_tl = *red*
il_tl = *green*
ml_tl = *green*
⊢
1 = 0 —EQ_LR—
green ≠ *red*
green = *red*
il_tl = *green*
⊢
1 = 0
—NOT_L, HYP—

...
—OR_L—
green ≠ *red*
il_tl = *red*
il_tl = *green*
ml_tl = *green*
—EQ_LR—
green ≠ *red*
green = *red*
ml_tl = *green*
⊢
1 = 0
—NOT_L, HYP—

イベント ML_tl_green **と** IL_tl_green **を変更する**

この新しい不変式 **inv2_5** は 6.3 節で提案したイベント ML_tl_green と IL_tl_green では明らかに保存されない。そのため、以下のように[*18]別の信号機を赤に変えるようにする。

ML_tl_green
when
ml_tl = *red*
a + *b* < *d*
c = 0
then
ml_tl := *green*
il_tl := *red* [New!]
end

IL_tl_green
when
il_tl = *red*
0 < *b*
a = 0
then
il_tl := *green*
ml_tl := *red* [New!]
end

イベント ML_out **が不変式 inv2_3 を保存することを証明する**

イベント ML_out が不変式 **inv2_3** を保存することを証明しようとすると、またしても厄介事に巻き込まれる。以下が証明義務である。

[*18] 訳注: 記号 [New!] は原文では下線です。

...	...	
inv2_3	$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$	
inv2_4	$il\_tl = green \Rightarrow 0 < b \wedge a = 0$	ML_out / **inv2_3** / INV
ML_out の限定条件	$ml\_tl = green$	
⊢	⊢	
変更後の **inv2_3**	$ml\_tl = green \Rightarrow a + 1 + b < d \wedge c = 0$	

以下は仮の証明である (MON 適用後)。

$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
⊢
$ml\_tl = green \Rightarrow a + 1 + b < d \wedge c = 0$
—IMP_R—
$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$
$ml\_tl = green$
⊢
$a + 1 + b < d \wedge c = 0$
—IMP_L—
$a + b < d \wedge c = 0$
$ml\_tl = green$
⊢
$a + 1 + b < d \wedge c = 0$
—AND_L—
$a + b < d$
$c = 0$
$ml\_tl = green$
⊢
$a + 1 + b \wedge c = 0$
—AND_R—
...

...	...
—AND_R—	—AND_R—
$a + b < d$	$a + b < d$
$c = 0$	$c = 0$
$ml\_tl = green$	$ml\_tl = green$
⊢	⊢
$a + 1 + b < d$	$c = 0$
—?—	—MON—
	$c = 0$
	⊢
	$c = 0$
	—HYP—

見て分かるように、2 個のシークエントのうちの前者は、$a + 1 + b$ が d に等しいとき ml_tl が赤になるのでなければ証明できない。この状況は、橋に入ることができる最後の 1 台の自動車が、この時点で入ってくる

ことを意味する。要求 FUN-3 は d を超える自動車が島と橋にいてはいけないことを言っている。このことから、イベント ML_out は以下のように 2 個のイベントに分割される。(ただし、両方のイベントが共通の抽象イベントの精義化である)

ML_out_1
when
$ml\_tl = green$
$a + b + 1 \neq d$ [New!]
then
$a := a + 1$
end

ML_out_2
when
$ml\_tl = green$
$a + b + 1 = d$ [New!]
then
$a := a + 1$
$ml\_tl := red$ [New!]
end

イベント IL_out が不変式 inv2_4 を保存することを証明する

同じ理由で、b が 1 のときイベント IL_out は不変式 **inv2_4** を保存しない。この場合、最後の自動車が島を離れる。結論としては、島の信号機は赤に変わるべきである。前節[*19]での ML_out と同じように、イベント IL_out を分割する必要がある。

IL_out_1
when
$il\_tl = green$
$b \neq 1$ [New!]
then
$b, c := b - 1, c + 1$
end

IL_out_2
when
$il\_tl = green$
$b = 1$ [New!]
then
$b, c := b - 1, c + 1$
$il\_tl := red$ [New!]
end

6.8 新しいイベントの収束

いまや私たちは新しいイベントが発散しないことを証明しなくてはならない。そのためには、全ての新しいイベントで減小する変量を明示する必要がある。実を言うと、それは不可能であることが判明する。簡単に言うと、a と c がともに 0 に等しい、すなわち橋の上に自動車がないとき、信号機は自由かつ永久に色を変えることができる。それは新しいイベント ML_tl_green と IL_tl_green を見れば分かる (再掲)

[*19] 訳注: 「前節」とは第 2 章、6 節、6.7 節よりもさらに下位の、番号のない節のこと。

ML_tl_green
when
ml_tl = *red*
$a + b < d$
$c = 0$
then
ml_tl := *green*
il_tl := *red*
end

IL_tl_green
when
il_tl = *red*
$0 < b$
$a = 0$
then
il_tl := *green*
ml_tl := *red*
end

実際に起こりうるのは、運転手が決して通れないほど速く信号が変わり続けることである[*20]。私たちは信号機の色が変わる方法をより統制されたものにする必要がある。すなわち、自動車が橋を通っているときのみ信号を変えるものとする。そのために、2 個の変数 *ml_pass* と *il_pass* を追加する。これらの変数は 2 個の値 TRUE と FALSE を取る。TRUE と FALSE は組込みの集合 BOOL の要素である。*ml_pass* が TRUE のとき、本土側の信号機が青に変わってから 1 台以上の自動車が島に向けて走っていることを意味する。*il_pass* が TRUE のときについても同様である。これらの変数は以下の不変式で定式化できる。

変数: ...
ml_pass,
il_pass

inv2_6: *ml_pass* ∈ BOLL
inv2_7: *il_pass* ∈ BOOL

ここで私たちはイベント ML_out_1, ML_out_2, IL_out_1, IL_out_2 を *ml_pass* または *il_pass* を TRUE に設定するように変更する必要がある。

ML_out_1
when
ml_tl = *green*
$a + b + 1 \neq d$
then
$a := a + 1$
ml_pass := TRUE [New!]
end

ML_out_2
when
ml_tl = *green*
$a + b + 1 = d$
then
$a := a + 1$
ml_tl := *red*
ml_pass := TRUE
end

IL_out_1
when
il_tl = *green*
$b \neq 1$
then
$b := b - 1$
$c := c + 1$
il_pass := TRUE
end

IL_out_2
when
il_tl = *green*
$b = 1$
then
$b := b - 1$
$c := c + 1$
il_tl := *red*
il_pass := TRUE
end

[*20] 訳注: 収束性の証明義務を理解するために有益な比喩だと思います。

さらにイベント ML_tl_green と IL_tl_green は、*ml_pass* と *il_pass* を FALSE にリセットするように変更し、また限定条件に条件 *il_pass* = TRUE と *ml_pass* = TRUE をそれぞれ追加する。これは以下のようになる。

ML_tl_green
when
ml_tl = *red*
a + *b* < *d*
c = 0
il_pass = TRUE [New!]
then
ml_tl := *green*
il_tl := *red*
ml_pass := FALSE[*New*!]
end

IL_tl_green
when
il_tl = *red*
0 < *b*
a = 0
ml_pass = TRUE [New!]
then
il_tl := *green*
ml_tl := *red*
il_pass := FALSE
end

これらの変更を全て終えると、新しいイベントが発散しないことを保証するために証明すべきものを言明することができる。まず、私たちが明示する変量は以下。

variant_2: *ml_pass* + *il_pass*

しかしながら、変数 *ml_pass* と *il_pass* が真偽値であって自然数ではないことを考えると、この変量は正しくない。これを正すには、真偽値の式を自然数の式に変換すればよい。これは、集合 BOOL から集合 {0, 1} への写像である定数 *b_2_n* によって直接的に行われる。

定数: ...
b_2_n

axm2_3: $b\_2\_n \in \mathrm{BOOL} \to \{0, 1\}$
axm2_4: *b_2_n*(TRUE) = 1
axm2_5[*21]: *b_2_n*(FALSE) = 0

前掲の変量は以下のように正確に定義できる。

variant_2: *b_2_n*(*ml_pass*) + *b_2_n*(*il_pass*)

イベント ML_tl_green と IL_tl_green に証明義務規則 VAR を適用した結果は以下のシークエントである。

ml_tl = *red*
a + *b* < *d*
c = 0
il_pass = TRUE
⊢
b_2_n(*il_pass*) < *b_2_n*(*ml_pass*) + *b_2_n*(*il_pass*)

il_tl = *red*
b > 0
a = 0
ml_pass = TRUE
⊢
b_2_n(*ml_pass*) < *b_2_n*(*ml_pass*) + *b_2_n*(*il_pass*)

[*21] 訳注: 原文では **axm2_4** が 2 個ありますが、訳文では 2 個目を **axm2_5** に直しています。

この時点では、第 1 のシークエントでは *ml_pass* = TRUE、第 2 のシークエントでは *il_pass* = TRUE がないと証明できない。これは以下の不変式を追加する必要性を示唆する。

inv2_8: $ml\_tl = red \Rightarrow ml\_pass = \text{TRUE}$
inv2_9: $il\_tl = red \Rightarrow il\_pass = \text{TRUE}$

この新しい不変式 **inv2_8** と **inv2_9** が全てのイベントで保存されることを証明しなければならない。これは読者の練習問題とする。

6.9 相対的無デッドロック

いま相対的無デッドロック性の証明義務規則 DLF の証明が残っている。「相対的」無デッドロックが私たちの例では「絶対的」無デッドロックにもなるのは、以前の抽象モデルで「絶対的」無デッドロック性が証明されているからである。証明すべき言明は、単純化された前提 (全ての不変式は必要ではない) から限定条件の論理和へのシークエントである。

$d \in \mathbb{N}$
$0 < d$
$ml\_tl \in COLOR$
$il\_tl \in COLOR$
$ml\_pass \in \text{BOOL}$
$il\_pass \in \text{BOOL}$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$ml\_tl = red \Rightarrow ml\_pass = \text{TRUE}$
$il\_tl = red \Rightarrow il\_pass = \text{TRUE}$
$\vdash$
$(ml\_tl = red \land a + b < d \land c = 0 \land ml\_pass = \text{TRUE} \land il\_pass = \text{TRUE}) \lor (il\_tl = red \land a = 0 \land b > 0 \land ml\_pass = \text{TRUE} \land il\_pass = \text{TRUE}) \lor ml\_tl = green \lor il\_tl = green \lor a > 0 \lor c > 0$ [DLF]

以下は証明のスケッチである。このスケッチでは、たくさんの中間のステップを略し[*22]、シンボル—!—でステップの省略を表す。

$d \in \mathbb{N}$
$0 < d$
$b \in \mathbb{N}$
$ml\_tl = red$
$il\_tl = red$
$ml\_tl = red \Rightarrow ml\_pass = \text{TRUE}$
$il\_tl = red \Rightarrow il\_pass = \text{TRUE}$
$\vdash$
$(b < d \land ml\_pass = \text{TRUE} \land il\_pass = \text{TRUE}) \lor (b > 0 \land ml\_pass = \text{TRUE} \land il\_pass = \text{TRUE})$
—!—

[*22] 訳注: 原文ではシークエント計算は左から右に進むので、ステップの省略は右向きの波矢印で書かれています。訳文ではシークエント計算が上から下に進むので、記号—!—で行を区切っています。

$d \in \mathbb{N}$
$0 < d$
$b \in \mathbb{N}$
$ml\_tl = red$
$il\_tl = red$
$ml\_pass = \mathrm{TRUE}$
$il\_pass = \mathrm{TRUE}$
⊢
$(b < d \land ml\_pass = \mathrm{TRUE} \land il\_pass = \mathrm{TRUE}) \lor (b > 0 \land ml\_pass = \mathrm{TRUE} \land il\_pass = \mathrm{TRUE})$
—!—
$0 < d$
$b \in \mathbb{N}$
⊢
$b < d \lor b > 0$
—OR_R1—
$0 < d$
$b = 0$
⊢
$b < d$
—EQ_LR—
$0 < d \vdash 0 < d$
—HYP—

6.10 第 2 次精細化のまとめ

精細化を通して、証明が (というよりは証明の失敗が) 誤りを正し、モデルを改良するのに役立つのを見てきた。事実、私たちは 4 個の誤りを発見し、不変式を追加し、4 個のイベントを直し、2 個の変数を追加した。これが第 2 次精細化の最終版である。

変数: ...
- ml_tl
- il_tl
- ml_pass
- il_pass

inv2_1: $ml\_tl \in COLOR$
inv2_2: $il\_tl \in COLOR$
inv2_3: $ml\_tl = green \Rightarrow a + b < dc = 0$
inv2_4: $il\_tl = green \Rightarrow 0 < ba = 0$
inv2_5: $ml\_tl = red \lor il\_tl = red$
inv2_6: $ml\_pass \in \mathrm{BOOL}$
inv2_7: $il\_pass \in \mathrm{BOOL}$
inv2_8: $ml\_tl = red \Rightarrow ml\_pass = \mathrm{TRUE}$
inv2_9: $il\_tl = red \Rightarrow il\_pass = \mathrm{TRUE}$

variant_2: $b\_2\_n(ml\_pass) + b\_2\_n(il\_pass)$

以下は第 2 次精細化のイベントである。

ML_out_1
when
$ml\_tl = green$
$a + b + 1 \neq d$
then
$a := a + 1$
$ml\_pass :=$ TRUE
end

ML_out_2
when
$ml\_tl = green$
$a + b + 1 = d$
then
$a := a + 1$
$ml\_tl := red$
$ml\_pass :=$ TRUE
end

IL_out_1
when
$il\_tl = green$
$b \neq 1$
then
$b := b - 1$
$c := c - 1$
$il\_pass :=$ TRUE
end

IL_out_2
when
$il\_tl = green$
$b = 1$
then
$b := b - 1$
$c := c + 1$
$il\_tl := red$
$il\_pass :=$ TRUE
end

ML_in
when
$0 < c$
then
$c := c - 1$
end

IL_in
when
$0 < a$
then
$a := a - 1$
$b := b + 1$
end

ML_tl_green
when
$ml\_tl = red$
$a + b < d$
$c = 0$
$il\_pass =$ TRUE
then
$ml\_tl := green$
$il\_tl := red$
$ml\_pass :=$ FALSE
end

IL_tl_green
when
$il\_tl = red$
$0 < b$
$a = 0$
$ml\_pass =$ TRUE
then
$il\_tl := green$
$ml\_tl := red$
$il\_pass :=$ FALSE
end

7　第 3 次精義化: 自動車センサーの導入

7.1　序論

センサー

この精義化ではセンサーを導入する。センサーは橋に出入りする自動車の存在を感知する。センサーは道路の両側かつ端の両端に置かれていることを思い出しておく。これらを図 8 (訳文では省略) に示す。

コントローラーと環境の閉じたモデル

センサーの存在によって、***ソフトウェアによるコントローラー*** *(software controller)* と ***物理的な環境*** *(physical environment)* との分離が明確になる。これは図 9 (訳文では省略) のダイヤグラムで見ることができる。

見て分かるように、コントローラーは 2 個の ***チャネル*** *(channels)* を備えている。出力チャネルはコントローラーを信号機に繋いでおり、入力チャネルはセンサーをコントローラーに繋いでいる。私たちの目的は、コントローラーと環境の組み合わせの数学的な状態を表現する ***閉じたモデル*** *(closed model)* を作ることである。このようなモデルを作る理由は、コントローラーが環境との完全な調和のもとに動く——ただし、もちろん、環境がたくさんの前提に従うならば——ことを保証するためである[*23]。私たちはこれから、閉じたモデルの様々な構成要素を作るのに役立つ変数を見ていく。

コントローラーの変数

ソフトウェア・コントローラーのモデルは既に多数の変数を持っている。変数 a, b, c は自動車の数、そのうち a, c は橋の上、b は島の自動車の数である。真偽値の変数 *il_pass* と *ml_pass* は前節で導入された。重要なことは、変数 a, b, c は物理的な自動車の数を正確には反映していないことである。実際のところ、コントローラーは環境についての近似の絵によって動いている。しかし、それにもかかわらず、環境を正しく制御できることを証明したい。これがモデル構築プロセスの心臓である。

環境の変数

環境はセンサーの状態を表す 4 個の変数で定式化される。詳しく言うと、センサーは “on” または “off” の二者択一の状態を取る。自動車が上にいるとき “on” であり、そうでなければ “off” である。結論として、私たちはモデルの状態を以下の 4 個の変数で拡張する: *ML_OUT_SR*, *ML_IN_SR*, *IL_OUT_SR*, *IL_IN_SER*。これらの変数が大文字で命名されていることを注意しておく。これは ***物理的な変数*** *(phisical variables)*、すなわち、リアル世界の物体を表すものである。私たちはさらに 3 個の変数 A, B, C を導入する。これらは自動車の ***物理的な*** *(physical)* 数——A は橋の上の島に向かっている自動車の数、B は島にいる自動車の数、C は橋の上の本土に向かっている自動車の数——である。

出力チャネル

これから私たちはコントローラーと環境がどのように通信するかを表現する。私たちは既に変数 *ml_tl* と *il_tl* を導入した。これらはコントローラーから環境への出力チャネルを表現している。ここでは物理的な信号機は直接にこれらの変数で表されると前提とする。コントローラーがチャネルの色を変えてから実際に信号機

[*23] 環境が従うべき前提は 7.2 節で詳細に定義する。

の色が変わるまでに (非常に) 小さな遅れがあることを想像してもよいが、その遅れは無視できると考えると、この前提が導かれる。

入力チャネル

ここまで来て残っているのは、センサーがコントローラーと通信する方法を表現することである。センサーについての技術的な詳細には興味はなく、ただ外から見た振る舞いが分かればよい。既に述べたように、センサーは “on” と “off” の 2 種類の状態を取る。センサーが “off” から “on” に変化するとき、自動車がセンサーの上に来たことを検知したことを意味する。しかし、このときコントローラーには何も送らなくて良い。センサーはある程度の時間 (自動車が赤信号で待っている間) “on” のままであることを注意しておく。センサーの状態が “on” から “off” に変化するとき、センサーの上にいた自動車が離れていくことを意味する。このときにはコントローラーにメッセージが送られる。このことは図 10 (訳文では省略) のダイアグラムに示した。

これに伴い、私たちは 4 個の入力チャネル変数をそれぞれのセンサーについて用意する: *ml_out_10*, *ml_in_10*, *il_in_10*, *il_out_10*。

まとめ

以下は、この節で説明したそれぞれの種類の変数のまとめである。

入力チャネル: *ml_out_10*, *ml_in_10*, *il_in_10*, *il_out_10*
コントローラー: a, b, c, *ml_pass*, *il_pass*
出力チャネル: *ml_tl*, *il_tl*
環境: A, B, C, *ML_OUT_SR*, *ML_IN_SR*, *IL_OUT_SR*, *IL_IN_SR*

図 11 (訳文では省略) のダイアグラムは、閉じたモデルのさまざまな種類の変数を示している。原理的に、入力チャネル変数は環境がセットしコントローラーがテストする。同じように、出力チャネル変数はコントローラーがセットし環境がテストする。しかし、7.3 節できちんと説明するが、この例ではこの法則の例外がある。また、コントローラー変数はコントローラーがセットもテストも行い、環境変数は環境がセットもテストも行う。これについては例外はない。

7.2 状態の精義化

まず、センサーの状態 (“on” と “off”) を定義するキャリアセットを追加する。これは以下のようになる。

集合: . . . , *SENSOR*
定数: . . . , *on*, *off*

axm3_1: $SENSOR = \{on, off\}$
axm3_2: $on \neq off$

新しい変数と、その型を規定する基本的な不変式を以下に示す。

inv3_1: $ML\_OUT\_SR \in SENSOR$
inv3_2: $ML\_IN\_SR \in SENSOR$
inv3_3: $IL\_OUT\_SR \in SENSOR$
inv3_4: $IL\_IN\_SR \in SENSOR$
inv3_5: $A \in \mathbb{N}$
inv3_6: $B \in \mathbb{N}$

inv3_7: $C \in \mathbb{N}$
inv3_8: *ml_out_10* ∈ BOOL
inv3_9: *ml_in_10* ∈ BOOL
inv3_10: *il_out_10* ∈ BOOL
inv3_11: *il_in_10* ∈ BOOL

さらに、私たちはより興味深い不変式を導入する。まず、センサー *IL_IN_SR* が *on* のとき、*A* が正になることを明示する。言い換えれば、そのとき物理的な自動車が 1 台は橋の上にいて、そのうちの 1 台は具体的にはセンサー *IL_IN_SR* の上にいる。同じ不変式は *IL_OUT_SR* と *ML_IN_SR* にも適用でき、以下を得る。

3_12: *IL_IN_SR* = *on* ⇒ $A > 0$
3_13: *IL_OUT_SR* = *on* ⇒ $B > 0$
3_14: *ML_IN_SR* = *on* ⇒ $C > 0$

次に、入力チャネル *ml_out_10* が TRUE のとき、これは自動車がセンサー *ML_OUT* を離れたことを意味する。そして、これは本土の信号機が青のときのみ可能である。以下の不変式は自動車の運転手が信号を守ることを表す。なお、同じことが入力チャネル *il_out_10* についても可能である。これは以下の 2 個の不変式で定式化される。

3_15: *ml_out_10* = TRUE ⇒ *ml_tl* = *green*
3_16: *il_out_10* = TRUE ⇒ *il_tl* = *green*

次のグループの不変式はセンサーの状態とコントローラーへのメッセージの関係を扱うものである。すなわち、自動車がセンサーの上にいるときは入力チャネルにメッセージが送られないことを意味する。これらの不変式は以下。

3_17: *IL_IN_SR* = *on* ⇒ *il_in_10* = FALSE
3_18: *IL_OUT_SR* = *on* ⇒ *il_out_10* = FALSE
3_19: *ML_IN_SR* = *on* ⇒ *ml_in_10* = FALSE
3_20: *ML_OUT_SR* = *on* ⇒ *ml_out_10* = FALSE

自動車がセンサーの上にいるとき、センサーから送られた以前のメッセージは既にコントローラーに処理されていることを、これらの不変式は示している。これを満たすためには以下の 2 通りの解釈が考えられる。

(A) コントローラーが準備完了するまで、自動車はセンサーの手前で待つ。
(B) コントローラーは高速で、次の自動車が来るまでに必ず準備完了する。

明らかに、(A) は受け入れられない。しかるに、私たちは (B) を前提とする。もしそうでなければ、コントローラーはシステムに入るまたはシステムから離れる自動車のうちいくつかを見逃す。言い換えれば、***この前提はシステムを設置するときに確認しなくてはならない*** *(this assumption has to be checked when installing the system)*。すると、以下の要求が私たちの要求文書に欠けていたことが分かる。

コントローラーは、環境から来る全ての情報を処理できるほど高速である。[FUN-5]

不変式の次のグループは、自動車の物理的な数 (A, B, C) とコントローラーが認識している数 (a, b, c) の関係を示す。

3_21: *il_in_10* = TRUE ∧ *ml_out_10* = TRUE ⇒ $A = a$
3_22: *il_in_10* = FALSE ∧ *ml_out_10* = TRUE ⇒ $A = a + 1$
3_23: *il_in_10* = TRUE ∧ *ml_out_10* = FALSE ⇒ $A = a - 1$
3_24: *il_in_10* = FALSE ∧ *ml_out_10* = FALSE ⇒ $A = a$

これらの不変式を理解するのは簡単である。*il_in_10* = TRUE のとき、自動車は橋を離れ島に入ることを意味するが、コントローラーはこのことをまだ知らない。そのため、A は 1 増え B は 1 減るのに対して、a と b は変化しない。同様に、*ml_out_10* = TRUE のとき、自動車が本土から離れ橋に入ることを意味するが、コントローラーはこのことをまだ知らない。そのため、A は 1 増えるが a は変化しない。同じような不変式を B と b、C と c についても作り、以下のようになる。

3_25: *il_in_10* = TRUE ∧ *il_out_10* = TRUE ⇒ $B = b$
3_26: *il_in_10* = TRUE ∧ *il_out_10* = FALSE ⇒ $B = b + 1$
3_27: *il_in_10* = FALSE ∧ *il_out_10* = TRUE ⇒ $B = b - 1$
3_28: *il_in_10* = FALSE ∧ *il_out_10* = FALSE ⇒ $B = b$

3_29: *il_out_10* = TRUE ∧ *ml_in_10* = TRUE ⇒ $C = c$
3_30: *il_out_10* = TRUE ∧ *ml_in_10* = FALSE ⇒ $C = c + 1$
3_31: *il_out_10* = FALSE ∧ *ml_in_10* = TRUE ⇒ $C = c - 1$
3_32: *il_out_10* = FALSE ∧ *ml_in_10* = FALSE ⇒ $C = c$

最後の 2 個の——おそらく最も重要な——不変式は、***自動車の物理的な数が保持する*** *(hold for the physical number of cars)* 2 個の主要な性質 (片側通行の橋と自動車の数の制限) である。

3_33: $A = 0 \vee C = 0$
3_34: $A + B + C \leq d$

別の言い方をすれば、コントローラーは A, B, C についてのわずかに時間のずれた情報 (コントローラーは a, b, c を使って決定する) によって動いているにもかかわらず、物理的な数 A, B, C の基本的な性質を維持することができる。

7.3 コントローラーの抽象イベントの精義化

ここまで来れば抽象イベントの精義化は簡単である。これは以下のように一本道に行われる。

ML_out_1
when
ml_out_10 = TRUE [New!]
$a + b + 1 \neq d$
then
$a := a + 1$
ml_pass := TRUE
ml_out_10 := FALSE [New!]
end

ML_out_2
when
ml_out_10 = TRUE [New!]
$a + b + 1 = d$
then
$a := a + 1$
ml_tl := *red*
ml_pass := TRUE
ml_out_10 := FALSE [New!]
end

IL_out_1
when
il_out_10 := TRUE [New!]
$b \neq 1$
then
$b := b - 1$
$c := c - 1$
il_pass := TRUE
il_out_10 := FALSE [New!]
end

IL_out_2
when
il_out_10 := TRUE [New!]
$b = 1$
then
$b := b - 1$
$c := c + 1$
il_tl := *red*
il_pass := TRUE
il_out_10 := FALSE
end

これらのイベントの抽象版は信号機の色が青であることをテストしていたことを注意しておく。精義化後のイベントではその必要はない。なぜなら、これらのイベントの具象版は入力チャネル *ml_out_10* と *il_out_10* で起動されており、それらの入力チャネルは不変式 **inv3_15** と **inv3_16** で信号が青であることを保証しているからである。

ML_in
when
ml_in_10 = TRUE [New!]
$c > 0$
then
$c := c - 1$
ml_in_10 := FALSE [New!]
end

IL_in
when
il_in_10 = TRUE [New!]
$a > 0$
then
$a := a - 1$
$b := b + 1$
il_in_10 := FALSE [New!]
end

これらの 6 個のイベントは入力チャネルで起動され、そのチャネルをリセットすることが分かる。チャネルをリセットすることで、コントローラーの操作が終了したことを表している。次の節で説明するイベント xxx_arr は入力チャネルをセットする。それは次の自動車がセンサーの上に来るのを「許す」ためである。この相互作用は、コントローラーの反応が自動車の到着よりも速いことを表現するための形式的な方法である！

ML_tl_green
when
ml_tl = *red*
$a + b < d$
$c = 0$
il_pass = TRUE
il_out_10 = FALSE [New!]
then
ml_tl := *green*
il_tl := *red*
ml_pass := FALSE
end

IL_tl_green
when
il_tl = *red*
$a = 0$
ml_pass = TRUE
ml_out_10 = FALSE [New!]
then
il_tl := *green*
ml_tl := *red*
il_pass := FALSE
end

イベント *ML_tl_green* の新しい限定条件 *il_out_10* = FALSE は以下の不変式 **inv3_16** の保存に必要である。

inv3_16: *il_out_10* = TRUE ⇒ *il_tl* = *green*

これはイベント IL_tl_green で *il_tl* が *green* に設定されるからである。イベント ML_tl_green にも同じような限定条件 (*ml_out_10* = FALSE) がある。これは不変式 **inv3_16** を保存するためである。

いま挙げた 2 個のイベントに別の限定条件を付け加えることも考えられる。たとえば、自動車が通ろうとしているときのみ信号を青にするなど。そのためには、信号機の近くに追加の 2 個のセンサーを加えなければならない。このようなシステムの拡張は読者に委ねる。

7.4 環境に新しいイベントを追加する

自動車がセンサーに到着することに相当する 4 個の新しいイベントを追加する。

ML_out_arr
when
ML_OUT_SR = *off*
ml_out_10 = FALSE
then
ML_OUT_SR := *on*
end

ML_in_arr
when
ML_IN_SR = *off*
ml_in_10 = FALSE
$C > 0$
then
ML_IN_SR := *on*
end

IL_in_arr
when
IL_IN_SR = *off*
il_in_10 = FALSE
$A > 0$
then
IL_IN_SR := *on*
end

IL_out_arr
when
IL_OUT_SR = *off*
il_out_10 = FALSE
$B > 0$
then
IL_OUT_SR := *on*
end

それぞれのイベントで、以前のメッセージは既に処理されていることを前提としている。そのため、入力チャネルは FALSE であることをテストしている。さらに、自動車の物理的な数を前述のようにテストしている。このことは、センサーが “on” になるのは自動車が存在するためだという事実を表現している。これは要求 ENV-5——センサーは、橋に入るまたは橋から出る自動車の存在を検出するために使われる——に適合する。私たちは最後に、自動車がセンサーから離れることを表す以下の 4 個のイベントを得る。

ML_out_dep
when
ML_OUT_SR = *on*
ml_tl = *green*
then
ML_OUT_SR := *off*
ml_out_10 := TRUE
end

ML_in_dep
when
ML_IN_SR = *on*
then
ML_IN_SR := *off*
ml_in_10 := TRUE
$C := C - 1$
end

IL_in_dep
when
IL_IN_SR = *on*
then
IL_IN_SR := *off*
il_in_10 := TRUE
$A := A - 1$
$B := B + 1$
end

IL_out_dep
when
IL_OUT_SR = *on*
$il_t l$ = *green*
then
IL_OUT_SR := *off*
il_out_10 := TRUE
$B := B - 1$
$C := C - 1$
end

重要な注意点として、本土から出るセンサーから自動車が離れるのは対応する信号機の色が青のときである。同様に、島から出るセンサーから自動車が離れるのは対応する信号機の色が青のときである。ここで私たちは要求 ENV-3——自動車は信号が赤のとき通過せず、信号が青のときのみ通過する——を考えに入れている。ならびに、これらのイベントはコントローラーにメッセージを送っていると考えることもできる。最後に、自動車の物理的な数も期待されたように変化する。これによって、私たちは環境に起こることを模式化している。

7.5 新しいイベントの収束

全ての新しいイベントで減小する変量を私たちは明示する必要がある。

variant_3: $12-(ML\_OUT\_SR+ML\_IN\_SR+IL\_OUT\_SR+IL\_IN\_SR+2*(ml\_out\_10+ml\_in\_10+il\_out\_10+il\_in\_10))$

変量 **variant_2** と同じように、この変量はこのままでは正しくない。真偽値とセンサーの式を数値の式にしなければならない。しかし、それは読者に委ねる。

7.6 無デッドロック

この第 3 次精義化がデッドロックしないことの証明は読者に委ねる。