CPSの形式検証——超準解析によるアプローチ

蓮尾 一郎 （東京大学 情報理工学系研究科），末永 幸平 （京都大学 情報学研究科）

2014年8月

図 1: ハイブリッドシステムの例．加速度 a_0 で直線運動する列車が，$z = m$ の位置にあるカベに安全距離 s より近づくと加速度 $-b$ で減速する．

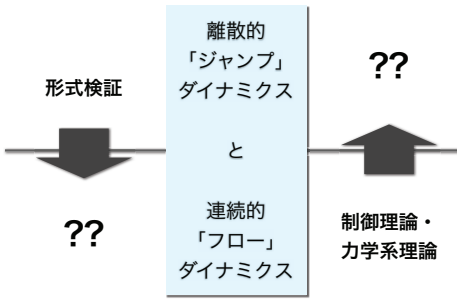

図 2: ハイブリッドシステム研究における 2 つのアプローチ．形式検証は元来，プログラム実行結果に代表される離散ダイナミクスを解析対象としてきた．その諸成果がハイブリッドシステムに応用されつつある．

1 はじめに

多くのサイバーフィジカルシステム（以下 CPS）は物理系としての連続的「フロー」と，計算機による離散的「ジャンプ」の 2 種類のダイナミクスを持ち，特にこの側面に注目した場合**ハイブリッドシステム**とよばれる．

最初に例を挙げておこう．正の定数 a_0, b, s のもとで図 1 のダイナミクスを考える．これは（位置 $z = m - s$ でのスイッチングにより）ハイブリッドシステムとなり，正確には次の微分方程式で表現される．

$$\begin{aligned} \dot{z} &= v \\ \dot{v} &= \begin{cases} a_0 & (m - z \geq s \text{ のとき}) \\ -b & (m - z < s \text{ のとき}) \end{cases} \end{aligned} \tag{1}$$

この例において「列車はカベにぶつからずに停止するか？」という自然な問題を考えよう．より正確に問題を述べると，

> 列車がカベにぶつからずに停止するために，
> 位置及び速度の初期値 z_0, v_0 がみたすべき
> 条件は？

となり，サイバーフィジカルシステムの品質保証問題の（初歩的な）一例になっている．この問いに答えるには，ダイナミクスに現れるフロー（等加速度運動）及びジャンプ（加速度のスイッチング）の両方を解析する必要がある．

ハイブリッドシステムは「離散，ジャンプ」と「連続，フロー」のハイブリッドであるから，その研究には大きく 2 つのアプローチ——離散からスタート，または，連続からスタート——が自然に存在する（図 2）．読者の多くの背景は制御理論であると思われるが，本解説はこのうち前者のアプローチ——フローから始める制御理論的アプローチとは逆の，ジャンプから始めるアプローチ——の解説である．特に本解説では，筆者らによる超準解析を用いた論理的アプローチ——数学的・論理的に厳密な形でフローをジャンプに変換する——に重点をおいて解説する．

```
1       n := N;   // := は変数への値の代入をあらわす
2       k := 1;
3       while (n > 0) {
4         k := k * n;
5         n := n - 1  }
```

図 3: プログラムの例．N の階乗 $N!$ を計算することは直感的には明らかだが，そのことを数学的に「証明」できるだろうか？

本解説の構成は以下のとおりである．まず，離散ダイナミクスの品質保証法としての**形式検証**とよばれるトピックについて，理論計算機科学の立場から概説する．実感をもってもらうため，特にプログラム論理によるプログラム検証を具体的に説明する．次に，形式検証を「ハイブリッド化」するためのいくつかの試みを紹介した後（これらは陽に微分方程式を導入する），著者らによる超準解析によるアプローチを述べる（ここではフローをジャンプに変換してしまうため，微分方程式があらわれない）．その数学的・論理的正当性を解説するための十分な紙幅はないが，最後に超準解析の入門も兼ねて概略の説明を試みる．

2 形式検証——プログラム論理を例に

図 3 のプログラムが N の階乗 $N! = N \cdot (N-1) \cdot \cdots \cdot 1$ を計算することは直感的には明らかであろう．しかしこのことに対して，(たとえば幾何学の証明のような）数学的な証明を与えることはできるだろうか？

図 3 のような小さな例はともかく，計算機が社会のあらゆる場面で重要な役割を果たす現代においては，設計した計算機システム——単体のプログラムのような小さな例から，多数のサーバが協調して並列に動作する大規模ネットワークまで——が期待通り動作するかという問題は，人命や大きな経済的損失に関わる非常に重要な問題である．

この問題（**計算機システムの品質保証**）に対し，現在産業界で多く用いられるのはは「いくつかの入力をテストケースとしてシステムを実行してみて，間違いがないかチェックする」という**テスト**の手法である．しかしながらテストではテストケース以外の入力に対しては「正しさの状況証拠」しか得られない．対照的に,「システムがあらゆる入力に対して正しく動作する」ことを**数学的に証明しよう**というのが**形式検証** *(formal verification)* という営みである [1]．

形式検証における数学的な議論及び証明のためには，登場人物——対象となるシステム，そのふるまい，及び要請される「正しさ」——をすべて形式的に記述する [2] 必要がある．特に「正しさ」が何を指すのかを形式的に記述したものを**仕様** *(specification)* とよぶ．

本節冒頭の例に戻ろう．図 3 のプログラムに対し，そのふるまいは明らかなものとして，仕様としては

プログラムの実行終了後の変数 `k` の値は $N!$ (2)

という性質を考えよう．この形式検証問題を解きたいのだが，ここではもう少し欲ばって，他の同種の形式検証問題にも適用可能であるような「スジのよい」やり方——実世界の巨大な計算機システムにもスケールするようなやり方——を追求したい．さらにすすめて証明を自動生成してくれるようなアルゴリズムができれば，計算機システムの品質保証における（人的）コストを劇的に圧縮することができる．これを可能にするのが，**ループ不変性質** *(loop invariant)* というアイデアであり，また**ホーア論理**というプログラム検証のための論理体系——**プログラム論理**——である．

3 ループ不変性質

図 3 のような `while` ループを含むプログラムの検証における最大の問題は，ループの中身の実行回数の見積もりの困難さである．図 3 では簡単であるが

[1] この文脈での「形式 formal」という言葉は，正確には「記号列の操作に基づいた」という意味だが，おおざっぱに「数学的な，数学的に正確な」と読みかえてもよい．

[2] 前注と同様,「数学的に定義する」の意．

一般にはとてもむずかしく，特にループ実行が停止するかどうか（**停止問題**）は計算機に判定できない**決定不能問題**の代表例になっている．

この際に用いられるのが**ループ不変性質**とよばれる概念である．ループ不変性質とは，ループの中身の実行の前後で保存される性質，すなわち

「ループの中身の実行の前に成立するならば，
実行後にも成立する」ことが保証されている　(3)

ような性質のことをよび，これによってループの中身の実行回数に依存しないプログラムの解析が可能になる．

さっそく例をあげよう．図 3 中のループに対しては，等式

$$\mathtt{k}\cdot(\mathtt{n}!) \;=\; \mathtt{N}! \tag{4}$$

がループ不変性質である．等式 (4) がループ不変性質になることはすぐに確かめられる： 実際，ループの中身の実行前・実行後の k と n の値をそれぞれ $k_{\mathrm{old}}, n_{\mathrm{old}}, k_{\mathrm{new}}, n_{\mathrm{new}}$ とし，実行前に等式 (4) が成立すると仮定すると，

$$k_{\mathrm{old}}\cdot(n_{\mathrm{old}}!) \;=\; \mathtt{N}! \qquad (\text{仮定より}) \tag{5}$$

$$k_{\mathrm{new}} \;=\; k_{\mathrm{old}}\cdot n_{\mathrm{old}} \qquad (\text{ループの中身より}) \tag{6}$$

$$n_{\mathrm{new}} \;=\; n_{\mathrm{old}} - 1 \qquad (\text{ループの中身より}) \tag{7}$$

がそれぞれ成立し，これらから次を得る．

$$k_{\mathrm{new}}\cdot(n_{\mathrm{new}}!) \overset{(6,7)}{=} (k_{\mathrm{old}}\cdot n_{\mathrm{old}})\cdot\big((n_{\mathrm{old}}-1)!\big) \overset{\text{簡単な計算}}{=} k_{\mathrm{old}}\cdot(n_{\mathrm{old}}!) \overset{(5)}{=} \mathtt{N}!$$

を得る．ゆえに等式 (4) はループ不変性質となる．

性質 A がループ不変性質であることの重要な帰結として，

ループの実行開始前（図 3 の 2 行目の後）に成立すれば，**ループの中身の実行回数にかかわらず**，ループの実行終了後（5 行目の後）でも成立する　(8)

という事実が得られる．この事実 (8) がループ不変性質の定義 (3) から従うことは明らかであろう．

最後に，ループ不変性質 (4) を用いて，図 3 のプログラムの仕様 (2) に対する検証——すなわち，仕様をみたすことの数学的証明——を行う．ループの実行開始前（2 行目の後）では $\mathtt{n}=\mathtt{N}$ かつ $\mathtt{k}=1$ ゆえ等式 (4) が成立する．等式 (4) はループ不変性質であるから，(8) より同じ等式がループの実行終了後（5 行目の後）でも成立する．ループの実行終了後にはループの実行条件 $\mathtt{n}>0$ が成立しないはずである——そうでなければループ実行が終了しない——ゆえに $\mathtt{n}=0$ が成立する [3]．よって等式 (4) において $\mathtt{n}=0$ とすると，$0!=1$ ゆえ $\mathtt{k}=\mathtt{N}!$ が成立する．証明終．

4　ホーア論理

読者は前節の証明を見てどのように感じられただろうか．ループ不変性質 (4) が天下り的に与えられたことをのぞけば [4]，議論は本質的に単純で機械的であるという印象を持たれたのではないだろうか．この機械的推論を，記号のパターンマッチと書き換えによる論理体系として整理したのが，計算機科学者 C.A.R. Hoare によって 1969 年に提案された**ホーア論理** *(Hoare logic)* である．

ホーア論理の系統だった記述は教科書 [4] などにゆずるとして，まず前節の証明をホーア論理で形式化したものを見てみよう（図 4）．この例を用いてホーア論理の形式的証明について説明する．

- ホーア論理の形式的証明は，各頂点に**ホーアトリプル** *(Hoare triple)* $\{A\}\,c\,\{B\}$ があり，各辺が**導出規則**の適用であるような**木**である．たとえば図 4 の形式的証明のもつ木構造は右の通り

[3] 変数の値が常に非負整数であることを暗黙の仮定として用いた．

[4] 実際，プログラム検証においてはループ不変性質の発見が最大の問題であり，数多くのヒューリスティックスが研究されている．

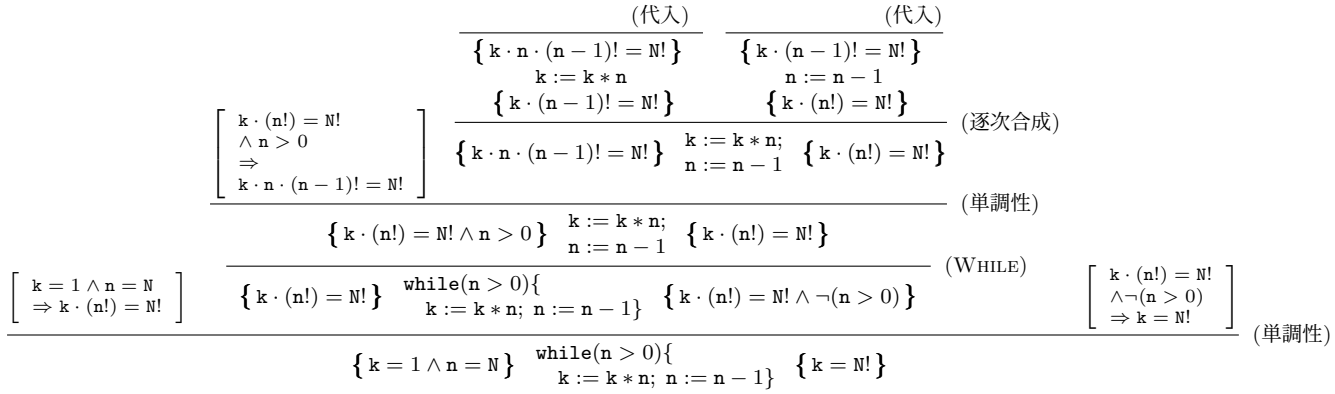

図 4: ホーア論理による形式的証明の例．各頂点にホーアトリプル $\{A\}\, c\, \{B\}$ があり，各辺が導出規則の適用であるような「木」になっている．

である．(ただし大括弧 $[\cdots]$ で囲まれた論理式は付帯条件であり，ここでは木の頂点としては数えない．後で (単調性) 規則と一緒に詳しく説明する．)

- **ホーアトリプル**とは，性質 A, B とプログラム c を中括弧で区切って書いた 3 つ組 $\{A\}\, c\, \{B\}$ のことをいう．その意味するところは

  > 性質 A が成立する状態からプログラム c を実行すると，停止後には性質 B が成立する

  であり，ゆえに A を**事前条件** *(precondition)*，B を**事後条件** *(postcondition)* とよぶ．たとえば

  $$\{\mathtt{n} = 3\}\ \mathtt{n} := \mathtt{n} + 1\ \{\mathtt{n} = 4\} \tag{9}$$

  は真であるホーアトリプルの例である．

- 図 4 の形式的証明では，導出規則を（機械的に）適用することで，真であるホーアトリプルを上から次々に導出して，最後に一番下の頂点（木の**根**）において目標とする言明——図 3 のプログラムの実行後に $\mathtt{k} = \mathtt{N}!$ となること——を導出している．

- **導出規則**の適用は横線 ———— であらわされ，その右側に記された (単調性)，(While)，(代入) などは適用された規則の名前をあらわす．

以下，ホーア論理の「キモ」であるところの導出規則について説明しよう．まず，図 4 で 2 回適用されている (単調性) 規則は次のようなものである．

$$\frac{A \Rightarrow A' \quad \{A'\}\, c\, \{B'\} \quad B' \Rightarrow B}{\{A\}\, c\, \{B\}}\ \text{(単調性)}$$

ここで $A \Rightarrow A'$ は「性質 A が A' を含意する」ことを意味する付帯条件であり [5]，プログラムの事前・事後条件について言明する（動的な）ホーアトリプルとは違い，**静的な**事実の言明である．図 4 で用いられる 3 つの付帯条件がすべて真であることは明らかであろう．

導出規則はすべて上から下に読む．ゆえに (単調性) 規則の直感的な意味は次のようになる．

> ホーアトリプル $\{A'\}\, c\, \{B'\}$ がすでに証明できていれば，その事前条件を強め，また事後条件を弱めた $\{A\}\ c\ \{B\}$ は真であると結論できる

次に，(逐次合成) 規則の一般形は

$$\frac{\{A\}\, c_1\, \{C\} \quad \{C\}\, c_2\, \{B\}}{\{A\}\, c_1; c_2\, \{B\}}\ \text{(逐次合成)}$$

であり，その正当性——上の 2 つのホーアトリプルがともに真であれば，下のホーアトリプルも真であ

[5] より正確には： 含意の論理式 $A \Rightarrow A'$ が恒真であるという条件．

る――は簡単に納得できるだろう．この規則の名前は，結論となるホーアトリプルがプログラム c_1, c_2 の逐次合成 $c_1; c_2$ に関するものであるという理由による．

2 か所の**葉**で用いられる (代入) 規則は，仮定を持たない（すなわち横線の上にホーアトリプルがない）規則である．

$$\frac{}{\{A[a/x]\}\ x := a\ \{A\}}\ (\text{代入})$$

ここで $A[a/x]$ は，性質（をあらわす記号列としての論理式）A において，変数 x を a に置き換えたものをあらわす．

この規則の正当性については少し考える必要があるだろう．たとえば A を $\mathtt{n} = 4$，x を $\mathtt{n}$，a を $\mathtt{n}+1$ とすると，次の (代入) 規則の適用が得られる．

$$\frac{}{\{\mathtt{n}+1 = 4\}\ \mathtt{n} := \mathtt{n}+1\ \{\mathtt{n} = 4\}}\ (\text{代入})$$

得られたホーアトリプルの事前条件は，実際 (9) のそれと同値である．すなわち，(9) が真であることを示すには，人間がアタマを使ってひねり出さなくても，実は記号の置き換えだけで十分だったのである．

最後に `while` ループに関する導出規則を述べる．

$$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \ \mathtt{while}(b)\{c\}\ \ \{A \wedge \neg b\}}\ (\textsc{While})$$

これはどういう意味だろうか？ 仮定であるホーアトリプルは A **がループ不変性質である**ことを主張している――その意味は (3) の条件ほぼそのままであり，事前条件にループ実行条件 b が加わることにより実は (3) の条件をリファインしている．そしてまた，結論となるホーアトリプルは (8) の条件ほぼそのままである（ループ実行が終了したということは，事後において b は偽のはず）．すなわち，上記のホーア論理の (While) 規則は 3 節で行った数学的議論の本質を導出規則として形式化したものである．

ホーア論理の導出規則を説明したところで，再度図 4 の型式証明を見てほしい．3 節の（形式的でない，自然言語による）証明がほぼそのまま形式的に木として表現されていることが見てとれるだろう．

一般にホーア論理によるプログラムの形式検証は次のようなワークフローをとる．

1. システム（ここではプログラム）と仕様を，ホーアトリプル $\{A\}\ c\ \{B\}$ として形式化する．
2. このホーアトリプルを根とする形式的証明（**証明木**）をつくる．これは木を下から上に生やしていくようなものであり，仮定がすべて（(代入) 規則のような仮定のない規則によって）閉じたら，それで成功である．人間が紙とペンで行うこともあれば，証明木を書くためのソフトウェア（**証明支援系** *(proof assistant)*，Coq や Isabell がよく知られている）の上で人間が行ったり，さらには全自動で証明を検索する**定理証明器** *(theorem prover)* を用いることもある．

このワークフローはさまざまな例に適用可能であり，このような**論理的検証**は，全数探索による**モデル検査** *(model checking)* とならびシステム検証の主要な手法の一つである．

5 離散ダイナミクスからハイブリッドシステムへ――微分方程式の導入

ここで 1 節に戻り，興味の対象をサイバーフィジカルシステム――特にハイブリッドシステム――に戻す．

以上に見てきたホーア論理による形式検証手法が離散的「ジャンプ」ダイナミクスを対象とするのは明らかだろう．このような離散的な形式検証の諸手法をハイブリッドシステムに拡張するにあたり，自然なやり方は「微分方程式を『プログラム』として認める」ことであろう．たとえば

$$\mathtt{while}(v \geq 0)(\dot{v} = a_0; \dot{z} = v) \qquad (10)$$

というようにプログラム中に明示的に微分方程式を書けるようにすれば，システムに連続的「フロー」を導入することができる．この自然なアイデアにも

とづいてプログラム論理を拡張しハイブリッドシステム検証手法を得たのが Platzer らの最近の成果であり [2][6]，注目を集めている．

また，モデル検査による形式検証の手法のハイブリッドシステムへの拡張はより早くに行われ，モデル検査の基礎となる**オートマトン**の拡張——**ハイブリッド・オートマトン** *(hybrid automaton)*——の理論は 1990 年代にかなり整備された．ここでも基本的なアイデアは，離散ダイナミクスをあらわすオートマトンの各状態に微分方程式を書くことにより連続ダイナミクスを導入するというものである．

```
while (v > 0) {
  if (m - z >= s) {
    a = a0
  } else {
    a = -b
  };
  z := z + v * dt;
  v := v + a * dt
}
```

図 5: 超準解析的プログラムの例 c_{train}．式 (1) のダイナミクスの，超準解析的プログラムによる表現．無限小値をあらわす定数 `dt` を用いて，連続ダイナミクスを `while` ループとして表現する．

6 ハイブリッドシステムの形式検証： 超準解析を用いてフローをジャンプに

前節で紹介したような「微分方程式を陽に持ち込む」ハイブリッドシステムの解析手法は，形式検証におけるさまざまなテクニック——形式的証明の検索とループ不変性質生成，時相論理式による仕様記述，状態空間の抽象化，$\cdots$——を適用することで大きな成功を収めている．一方で (10) のように微分方程式によって言語を拡張すると，それに対応した形式検証手法の拡張が必要になることも明らかである．具体的にたとえば 4 節のホーア論理で言うと，`while` や逐次合成など，プログラムの構成要素のそれぞれに対応する導出規則があるように，(10) のような微分方程式に対しても新たに導出規則を考案し，その正当性を示す必要がある．

これとは対照的に，筆者らが最近の論文 [3, 1] で提案したアプローチは

> 「フロー」を（無限個の，それぞれは無限に小さい）「ジャンプ」に変換することで，ハイブリッドシステムに離散的形式検証手法を**文字通りそのまま**適用する

というものであり，離散的な形式検証手法——たとえば 4 節のホーア論理の導出規則のレパートリー——を何ら拡張することなくそのままの形で適用することを可能にする．本解説の残りで，この**超準解析的アプローチ**を概説する．

図 5 を見てみよう．これは 1 節に例示した列車のダイナミクス（(1) の微分方程式で定義される）の**超準解析的プログラム**による表現であり，値 `z` と `v` の連続的な変化が `while` ループによって表現されている．たとえば 7 行目の代入文は $(\mathtt{z}_{\mathrm{new}} - \mathtt{z}_{\mathrm{old}})/\mathtt{dt} = \mathtt{v}$ から導かれることに注意されたい．

読者の中には「`dt` を小さな値（たとえば 10^{-3}）にとって，離散化による微分方程式の数値近似をしようとしているだけではないか？」と思われた方もおられるかもしれない．本アプローチのポイントは `dt` が本当の**無限小値** *(infinitesimal)* を表現することであり，ゆえに図 5 は列車のダイナミクスの数値近似ではなく，正確な，**任意精度** *(exact)* **の**モデリングになっていることである．

ハイブリッドシステムの形式検証に対する超準解析的アプローチのワークフローは以下のとおりである．

1. 検証対象のハイブリッドシステムを，無限小をあらわす定数 `dt` を含むプログラムとしてモデリングする．

   例：1 節の列車に対する図 5 のプログラム c_{train}．

2. システムへの仕様をホーアトリプルとして表現

[6] Platzer はホーア論理でなく**ダイナミック論理** *(dynamic logic)* をハイブリッドシステムに拡張したのであるが，これら 2 つのプログラム論理のスタイルに本質的な違いはない．

する．

例：列車の初期位置，初期速度がそれぞれ z_0, v_0 であれば列車はカベにぶつからず停止する，という仕様を次のように表現：$\{\mathtt{z} = z_0 \wedge \mathtt{v} = v_0\}\ c_{\text{train}}\ \{\mathtt{z} < \mathtt{m}\}$

3. 得られたホーアトリプルに対し，4 節と全く同じ導出規則を用いて証明木をつくることを試みる．

4 節の（離散的な）形式検証の枠組は，

- `while` ループを含むプログラミング言語 **While**
- 事前・事後条件など，性質 (assertion) を記述するための述語論理 **Assn**
- ホーア論理の導出規則のレパートリー **Hoare**

の 3 つ組からなるが，ハイブリッドシステムのために用いる拡張 $(\mathbf{While}^{\mathtt{dt}}, \mathbf{Assn}^{\mathtt{dt}}, \mathbf{Hoare}^{\mathtt{dt}})$ は次のようになる．

- プログラミング言語 $\mathbf{While}^{\mathtt{dt}}$ は，**While** に新たな定数 `dt` を付け加えた（だけの）ものである．
- 述語論理 $\mathbf{Assn}^{\mathtt{dt}}$ も同様 [7]．
- 論理体系 $\mathbf{Hoare}^{\mathtt{dt}}$ の導出規則は，もとの **Hoare** のそれと**全く同じ**．

すなわち，微分方程式を陽に持ち込んだり，それに対する新たな導出規則を導入したりすることなく，定数 `dt` を加えるだけで，離散的枠組をハイブリッドシステムに応用可能としたのである．`dt` によって拡張された枠組はもとの枠組と同じ位「よい性質」をもつ [3]．すなわち**健全性** *(soundness)*——ホーア論理で導出されるホーアトリプルは確かに真である——及び**相対完全性** *(relative completeness)*——真であるホーアトリプルはすべてホーア論理で導出できる [8]——が，もとの離散的枠組と同様に成立する．

[7] 正確には $\mathbf{Assn}^{\mathtt{dt}}$ における量化子 $\forall, \exists$ は実数でなく超実数上を動く，という違いがある．この事情は移転原理（後述）と同じ．

[8] 「相対」完全性とよばれるのは，(単調性) 規則の付帯条件——すなわち論理式の含意 $A \Rightarrow B$ の真偽——が機械的に判定できないことによる．文献 [4] を参照されたい．

7 論理学の成果たる超準解析とその応用

ライプニッツらによる解析学の定式化は無限小値の概念に依っていたが，無限小値——他の任意の正の値よりも小さな正の値——の存在は実数の理論に矛盾をきたすため，解析学の定式化は ε-δ 論法によるものに改められた．これに対し A. Robinson は 1960 年代に**超準解析** *(nonstandard analysis)* の理論を樹立し，無限小値を実体として矛盾なく扱う枠組を与えた．以下本節では，(少し込み入るが) 超準解析のできるだけ直感的な解説を試みる．

超準解析においては実数全体及び無限小値などを含んだ**超実数** *(hyperreal)* の集合 ${}^*\mathbb{R}$ を考える．最初はおおざっぱに，超実数とは実数の無限列 $(a_0, a_1, \dots)$ のことと考える．(普通の) 実数 r は定数列 $(r, r, \dots)$ によって超実数とみなされ，超実数上の加法や乗法などは要素ごとに定義される：$(a_i)_i + (b_i)_i = (a_i + b_i)_i$．ポイントは超実数上の述語の定義であり，たとえば大小関係 $(a_i)_{i\in\mathbb{N}} < (b_i)_{i\in\mathbb{N}}$ は

$$(a_i)_i < (b_i)_i \tag{11}$$

$$\overset{\text{def.}}{\iff} \quad \text{「ほとんどすべての } i \text{ に対して」} \quad a_i < b_i \tag{12}$$

$$\impliedby \quad \text{有限個の } i \in \mathbb{N} \text{ を除いて} \quad a_i < b_i \tag{13}$$

と定義される．すると超実数 $(1, \frac{1}{2}, \frac{1}{3}, \dots)$ が無限小値であることが簡単にわかる：任意の正の実数 $r > 0$ は超実数としては $(r, r, \dots)$ であり，有限個の n を除いて常に $\frac{1}{n} < r$ となるから $(1, \frac{1}{2}, \frac{1}{3}, \dots) < (r, r, \dots)$ が成立するのである．

しかし，なぜ (13) の条件は必要十分でなく $\Leftarrow$ のみなのだろうか？ 超実数 $(1, -1, 1, -1, \dots)$ を考えると，これは (13) の定義のもとでは 0 よりも大きくも小さくもない．すると超実数は全順序集合をなさなくなり，何かと都合が悪い．そのために超準解析では，(12) の「ほとんどすべて」を**超フィルタ** *(ultrafilter)* $\mathcal{F}$ を用いて正確に定義する．この正確な定義のもとでは，超実数 $(1, -1, 1, -1, \dots)$ は実数 1 または -1

のいずれかに等しい[9]．また超実数の定義は，正確には実数の無限列の $\mathcal{F}$ による適切な同値類である．

以上のもとで，次の「超準解析の基本定理」が得られる．

移転原理 *(transfer principle)*　ある適当なクラスの述語論理式 A について，A が実数 $\mathbb{R}$ のもとで恒真であることと，A が超実数 ${}^*\mathbb{R}$ のもとで恒真であることは同値．

超実数が全順序集合をなすという上述の問題も，論理式 $\forall x, y.(x \leq y \vee y \leq x)$ を「移転」する——$\mathbb{R}$ で真なので，${}^*\mathbb{R}$ でも真である——ことですぐさま証明できる．このように超準解析は（形式）論理学の成果であり，6 節で紹介したホーア論理の超準解析拡張は「述語論理に対して成立する移転原理を，プログラム論理という動的な状況に拡張したもの」ととらえることもできる．

以下，6 節の内容に超準解析の視点から補足を加えておく．最初に，図 5 を「プログラム」とよんだがこれは実際には**実行できない**：`dt` は無限小値であるため，図 5 の `while` ループは有限回では停止しないのである．一方で，そのプログラムの実行結果を超準解析の言葉で**数学的に定義**することはでき（プログラムの**形式意味論** *(formal semantics)*），その（計算できないが定義は可能な）実行結果に照らして，正しいホーアトリプルがホーア論理により導出されることが（健全性定理の形で）保証されている，というわけである．今述べたことは不思議に思えるかもしれないが，

- （有限の）プログラムが生成する（無限の）ふるまいを検証する際に，
- （無限の）ふるまいそのものを相手にするのは（有限の寿命しかない）人間には不可能なので，
- もともとの（有限の）プログラムをもとに検証した上で，その結果が（無限の）ふるまいについて有効であることを（健全性定理として）数学的帰納法により示す

というのは本手法に限らず理論計算機科学の基本である．

[9]が，具体的にどちらに等しいかはふつうわからない：答えは超フィルタ $\mathcal{F}$ の選び方に依存するのだが，$\mathcal{F}$ は選択公理を用いて非構成的に定義されるのでその実体は知れないのである．

8　まとめ

産業的・社会的に重要性をいや増すサイバーフィジカルシステムの品質保証について，元来離散ダイナミクスを対象とする形式検証手法の応用を，ホーア論理を具体例として手短に解説した．連続的「フロー」を導入するために微分方程式がしばしば陽に用いられるが，超準解析を用いて離散的枠組を「論理的にアップグレード」することもできる．

筆者らの研究は今のところプログラム論理による形式検証のアップグレードにとどまるが，モデル検査など他の形式検証手法のアップグレードについても現在研究をすすめている——原則的には，述語論理でその理論を書き下せる形式検証手法はすべて移転原理によってアップグレード可能である．また超準解析アプローチにおいては，ダイナミクスすべてをジャンプとみなすことにより統一的な取り扱いとさまざまな離散的テクニックの応用が可能となるが，これは微分方程式の有用性を否定するものでは全くない．微分方程式に対する諸知見も同時に活用できるようなシームレスな枠組を確立することにより，サイバーフィジカルシステム研究における 2 つのアプローチ——制御理論・力学系理論と，形式検証——の一座建立をめざしたい．

筆者らの研究 [3, 1] は科研費 24680001，25730040 及び FIRST 合原最先端数理モデルプロジェクトの助成を受けた．

参考文献

[1] Ichiro Hasuo and Kohei Suenaga. Exercises in *Nonstandard Static Analysis* of hybrid systems. In Proc. *CAV*, 2012.

[2] André Platzer. *Logical Analysis of Hybrid Systems—Proving Theorems for Complex Dynamics.* Springer, 2010.

[3] Kohei Suenaga and Ichiro Hasuo. Programming with infinitesimals: A while-language for hybrid system modeling. In Proc. *ICALP* , 2011.

[4] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.