解　説

プログラムの検証*

五十嵐　滋**

1. プログラムの検証と人工知能との関り合い

1970年代の初め頃までは，プログラムの検証は全体として人工知能学の中に包含されるテーマであるとの誤解ないしは錯覚を抱いていた人が多かったようである．そのような錯覚が普及してしまった原因の一つは，プログラムの検証に関する最も有力な提唱者であった J. McCarthy が同時にまた人工知能学の余りにも高名な指導者でもあったことに求められるのであろう．

しかしながら，誤解はあくまでも正すとなれば，プログラムの検証は，それ自体としては人工知能学とは独立して成立している分野である．ただし，いうまでもなく，この両分野には重要な共通性ないしは共通部分があるわけで，この際，それについて多少整理して見たほうがよいのではないかと思われる．

第一の共通性は，記号化された論理を取扱うことである．人工知能学では，定理の自動証明や質問応答系やロボットの問題解決の場面で記号論理が中心的役割を果たしているが，プログラムの検証も記号論理ぬきではほとんど取扱うことができない．

第二の共通性は，プログラムを取扱うことである．プログラムの検証がプログラムを取扱うことは当然であるが，人工知能も2種類の極めて重大なプログラムを取扱っている．すなわち，人工知能そのものの一部または全部を形成しているプログラム(ソフトウェア)と，人工知能が問題解決のために自力で作り出すプログラムとの2種類にほかならない．後者では，ロボットが目標を達成するための行動の順序付けられたプランが典型的であるが，プログラムの自動合成のように野心的な研究では，当然に計算機が普通に実行するためのプログラムが作り出されるわけである．プログラムの自動合成そのものが全体として人工知能学の中に包み込まれている分野であると言い切ることは問題を残すと思われるけれども，少なくとも今日のところ，その度合いはプログラムの検証よりは著しく大きいといえよう．

第三の共通性は，このように，記号論理的表現という共通の方法と，プログラムという共通の対象を持つことから派生する雑多な事ごとにある．例えば，定理の証明のためのプログラムが，そのままプログラム検証システムの一部分としてはめ込まれた例があるように，同じソフトウェアや同じノウハウが使える場合が少なからずある（それに，今日の世界では未だに記号論理が駆使できる人間は非常に少ないので，勢い共通な人脈グループで両方の研究が推進されることにもなっている．しかしながら，これは必ずしもよいことばかりではないであろう）．

第四の共通性は，人工知能学もプログラムの検証もともに，知能あるいはプログラミングというように，極めて高度な知的活動を追求していることにある．したがって，上述した多くの技術的な共通性と相まって，両者をあくまでも識別することに無理が生ずる場面もまたありうるわけである．冒頭あえて「錯覚」と呼んだのは，プログラムの検証がすべて全く人工知能学の問題だと考えたときに，検証の技術的発展の方向に偏りが生ずることに対する警鐘として受取って頂ければよいと思う．

このように人工知能学とプログラムの検証とは，多大な共通部分を保ちつつ，情報学の最も本質的な問題を追求しているわけである．

本稿では，以下，普通の「シクェンシャル」なプログラムと，オペレーティング・システムの基本になる「パラレル」なプログラムで，検証理論の発展の際に例示されてよく知られているものを1つずつ選び，「インフォーマルな」(すなわち，形式的でない，厳格で

* Verification of Programs by Shigeru IGARASHI (Institute of Information Sciences, University of Tsukuba).
** 筑波大学電子情報工学系

ない，直観的な）論証と，それを記号化して，本質的には「フォーマルな証明」のレベルに達したともいえる論証を提示する．なお，ここで使っている手法は，IFIP WG 2.2 や Séminaire IRIA などで発表したもののように，一般に入手しにくい筆者のプレプリント類に基づいているものである（パラレルなプログラムについての議論の記号化について書くのは，今度がはじめてである）．

2. 表明の論証方法

プログラムが終了して，その計算結果がある条件を満足することを表現した一種の言明のことを（強い）表明という〔これに対し，終了について言及しないものを，弱い表明という〕．

R. Floyd が弱い表明を論証したときの例題を，使用する言語についての配慮から，少し表現を変えて扱うことにする．

数列 $(a_i)_{i=1,2,\cdots}$ の部分和 $\sum_{i=1}^{n} a_i$ を計算するプログラムを考える．数列は，自然数上の実数値関数にほかならないから，この数列を表すのに，あらためて f という関数の記号を導入して，$f(i)=a_i$ $(i=1,2,\cdots)$ と定義する．Σ は正確に考えると，2つの自然数と1つの自然数上の実数値関数に1つの実数を対応づける関数であるが，これはプログラムの中でなく，証明の中にしか現れないので，そのまま使うことにする．

プログラムは，いうまでもなく，

```
i←0;
x←0;
while i<n do
  begin i←i+1;
        x←x+f(i)
  end                                  (2.1)
```

と書ける．**図-1** はこのフローチャートである．

FORTRAN では(2.1)は，

```
      X=0
      DO 101 I=1,N
  101 X=X+F(I)
      STOP
      END                              (2.2)
```

となるし，ALGOL 63 では，**while** の直前に **for** を補い，さらに処理系によっては←を：＝で書き換えておく必要がある．

これらの，言語は異なるが，アルゴリズムとしては同一のプログラムに対する検証方法は，どれか一種のプログラム言語についてできていれば，類似のレパートリーを持つ言語については，単なる移し替えで済むことは明らかであるが，その一方ある一つの言語の特長または癖を徹底的に利用して書かれているプログラムの検証には，やはりその言語と処理系に基づいた技術的準備が必要となることも覚悟せねばならないであろう．

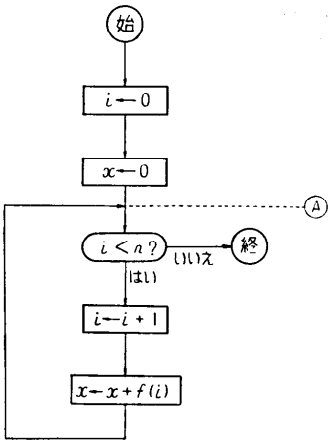

図-1　強い表明のための例題

さて (2.1) のプログラムが正しくできていることについては，いくらでも異なる証明方法が考えられる．その場合なるべく自然なやり方で，プログラムが終了することも同時に証明できるほうがよいと思われる．ここでは五十嵐 (1974) のト-論理の考え方を，まず，形式的なことを無視して，使って見ることとする．(2.1) のプログラムについて，次のように「論証」することは受容れられると思う．

A 1. まず while の所（つまりⒶ点で閉じているループ）に注目する．$i=n-k$ のとき，この while ステートメントを実行すると，それは必らず終了して，x の値が $\sum_{j=n-k+1}^{n} f(j)$ だけ増加することを k に関する数学的帰納法で証明する（A2～A3）．

A 2. $k=0$ のとき．$i=n$ であるから，本体に入らず直ちに終了．すなわち，x の値は $\sum_{j>n}^{n} f(j)=0$ だけ増加する．

A 3. $k+1$ のとき．$i=n-(k+1)<n$ であるから，少なくとも1回本体を実行する（つまりループを回る）．この結果 i は1増えるから，$i=n-k$ となり，x の値は $f(i)=f(n-k)$ だけ増加することは，本体

の簡単なステートメントから明らかである．ところで，$i=n-k$ であれば，これは帰納法の仮定を満たすから，while ステートメントはいずれ終了し，x の値はさらに $\sum_{j=n-k+1}^{n} f(j)$ だけ増加する．結局，最初の x からは $f(n-k)+\sum_{j=n-k+1}^{n} f(j)=\sum_{j=n-(k+1)+1}^{n} f(j)$ だけ増えて終了することになり，$k+1$ の場合が証明できた．

A 4. (2.1)全体に戻ると，while の前のステートメントで，$i=x=0$ となる．このとき $k=n$ であるから，A1 の結果を使えば，x は $\sum_{j=1}^{n} f(j)$ となって，全体が終了する．(論証終り)

3. 強い表明の記号化した証明

上の論証を記号化することを試みる．それには「終了する」ということを表す記号を導入しなければならないが，形式化の中間的段階として，ひとまず，

$$\mathcal{F}\{\mathrm{A}\}\vdash\mathcal{G} \tag{3.1}$$

という形の式を使うことにする．ここで，$\mathcal{F}$, $\mathcal{G}$ は命題の形で表記された，変数の間の関係式，Aはプログラム（またはその一部分）である．(3.1) は，「$\mathcal{F}$ が成立っているとき，Aを実行すると，必らず終了して，$\mathcal{G}$ が成立つ」と読むことにする．この記法を使って A1～A4 を少し論理を分析しながら書きなおすと次のようになる．

A 1. $x=a$ & $i=n-k$
{while $i<n$ **do begin**
$i\leftarrow i+1$; $x\leftarrow x+f(i)$ **end}**

$$\vdash x=\sum_{j=n-k+1}^{n} f(j) \tag{3.2}$$

を k に関する数学的帰納法で証明したい．ここに a は，x の初期値を表わすために導入された，プログラムとは無関係の変数である．[& は「かつ」]

A 2. $k=0$ のとき．$i=n-0=n$ だから，$i<n$ は否定される．すなわち，$i=n-0\supset\neg i<n$. [$\neg$ は否定，$\supset$ は「ならば」]

ところで任意の while ステートメントについて，次の法則性が成り立つことは明らかである．

プログラム中の変数の値の組が，関係式としてみたときに命題 $\mathcal{F}$ を満たしているならば，必らず $\mathcal{C}$ の否定が成立つとする．[すなわち $\mathcal{F}\supset\neg\mathcal{C}$] 変数がこのような値をとっているとき，**while** $\mathcal{C}$ **do** A は（定義によりなにもしないで終了するから）変数の値を変えない．それゆえ，このステートメントは終了し，かつ関係式 $\mathcal{F}$ は保存される．

これは，つぎのような「推論」の規則として把握し，採用することができる［いうまでもなく，推論とは，一つまたはそれ以上の「前提」である言明から，一つの新しい言明を「結論」として導きだすこと，およびその繰返し行為である］．

(R 1) $\mathcal{F}\supset\neg\mathcal{C}$ から $\mathcal{F}$**{while** $\mathcal{C}$ **do** A**}**$\vdash\mathcal{F}$ が導びかれる．

それゆえ，今の場合，

$$\begin{aligned}&x=a\ \&\ i=n-0\\&\{\textbf{while } i<n \textbf{ do}\cdots\}\\&\vdash x=a\ \&\ i=n-0.\end{aligned} \tag{3.3}$$

上式の最後の & $i=n-0$ は余分である．これは次の一般的な規則によって簡単化できる．

(R 2) $\mathcal{F}\{\mathrm{A}\}\vdash\mathcal{G}$ と $\mathcal{G}\supset\mathcal{H}$ から $\mathcal{F}\{\mathrm{A}\}\vdash\mathcal{H}$ が導びかれる．

今の場合，$(x=a\ \&\ i=n-0)\supset i=n-0$ で，これと (3.3)から，(3.3) の末尾の & 以下を切捨てたものが得られる．

A 3. ここで使われる論法は，結局のところ次の規則で表現されるものである．

(R 3) $\mathcal{F}\supset\mathcal{C}$, $\mathcal{F}\{\mathrm{A}\}\vdash\mathcal{G}$, および $\mathcal{G}$**{while** $\mathcal{C}$ **do** A**}**$\vdash\mathcal{H}$ から $\mathcal{F}$**{while** $\mathcal{C}$ **do** A**}**$\vdash\mathcal{H}$ が導びかれる．

すなわち，$\mathcal{F}$ として $x=a\ \&\ i=n-(k+1)$ を，$\mathcal{G}$ として $x=a+f(n-k)\ \&\ i=n-k$ を，$\mathcal{H}$ として $x=a+\sum_{j=n-(k+1)+1}^{n} f(j)$ をとり，$\mathcal{C}$ と A は考えている while ステートメントのとおりにとると，(R 3)の 3 つの前提から，必要な結論，すなわち

$$\begin{aligned}&x=a\ \&\ i=n-(k+1)\\&\{\textbf{while } i<n \textbf{ do begin}\\&\ i\leftarrow i+1;\ x+f(i) \textbf{ end}\}\\&\vdash x=a+\sum_{j=n-(k+1)+1}^{n} f(j)\end{aligned} \tag{3.4}$$

が導びかれるわけである．3 つの前提を順に確めると，最初のは自明であり，$\mathcal{F}\{\mathrm{A}\}\vdash\mathcal{F}$ は，次の公理および規則から導びかれる．

(Ax. 1) $\mathcal{F}[t]\{x\leftarrow t\}\vdash\mathcal{F}$.

ここに $\mathcal{F}[t]$ は，命題 $\mathcal{F}$ の中に現われている x を項［数式］で置きかえたものである．今，この公理によって，

$$x+f(i)=a+f(n-k)\ \&\ i=n-k$$

$\{x \leftarrow x+f(i)\}$

$\vdash x=a+f(n-k)\ \&\ i=n-k$ (3.5)

および

$x+f(i+1)=a+f(n-k)\ \&\ i+1=n-k$

$\{i \leftarrow i+1\}$

$\vdash x+f(i)=a+f(n-k)\ \&\ i=n-k$ (3.6)

が順次導びかれる．(3.5), (3.6)と，規則

(R 4) $\mathcal{F}\{A\}\vdash\mathcal{G}$ と $\mathcal{G}\{B\}\vdash\mathcal{H}$ から $\mathcal{F}\{A;B\}\vdash\mathcal{H}$ が導かれる．

とにより，

$x+f(i+1)=a+f(n-k)\ \&\ i+1=n-k$

$\{i \leftarrow i+1;\ x \leftarrow x+f(i)\}$

$\vdash x=a+f(n-k)\ \&\ i=n-k.$ (3.7)

前出の(R 2)の姉妹的規則で，

(R 5) $\mathcal{F}\supset\mathcal{G}$ と $\mathcal{G}\{A\}\vdash\mathcal{H}$ から $\mathcal{F}\{A\}\vdash\mathcal{H}$ が導かれる．

を使うと，

$x=a\ \&\ i=n-(k+1)$

$\supset x+f(i+1)=a+f(n-k)\ \&\ i+1=n-k$

[⊃は最後に読む]. (3.8)

と(3.7)から，

$x=a\ \&\ i=n-(k+1)$

$\{i \leftarrow i+1;\ x \leftarrow x+f(i)\}$

$\vdash x=a+f(n-k)\ \&\ i=n-k$ (3.9)

が得られる．最後の前提は，帰納法の仮定 (3.2) の a のところに $a+f(n-k)$ を代入した後，(R 2)で結果を修正すれば得られる．このような導出は，当り前のようであるが，実は厳格にやっておかないと矛盾が入りやすいところである．さしあたり，次のような規則にしておけば十分であろう．

(R 6) $\mathcal{F}[a]\{A\}\vdash\mathcal{G}[a]$ から $\mathcal{F}[t]\{A\}\vdash\mathcal{G}[t]$ が導かれる．ただし，a は A の中には現われていない変数で，t はAの中に現われている変数を全く含まない項に限られる*.

最後に，数学的帰納法を規則として述べておく必要がある．これも無制限に使うと矛盾を生ずるからである．

(R 7) k が自然数の値をとる変数で，Aの中には現われていないものとする．$\mathcal{F}[0]\{A\}\vdash\mathcal{G}[0]$ が得られ，かつ $\mathcal{F}[k]\{A\}\vdash\mathcal{G}[k]$ を仮定すれば $\mathcal{F}[k+1]\{A\}\vdash\mathcal{G}[k+1]$ が導かれるならば，$\mathcal{F}[k]\{A\}\vdash\mathcal{G}[k]$ が導かれる**.

* 制限がないと，たとえば $0=0\{x \leftarrow y\}\vdash x=y$ (正しい) から，$0=0\{x \leftarrow x+1\}\vdash x=x+1$ (誤り) がでて矛盾する．

** 仮定を伴う証明の中で (R6) を使うときは，a は仮定の中にも現れてはならない (反例省略).

(証明終り)

ここで簡単に統計をとって見ると，最初に記した厳格でない論証は約 20 行，後のほうの相当に厳格な論証は約 40 行で，しかもその中に問題の記述も含まれている．また公理および推論規則の記述と説明が約50行になっている．プログラムは6行で，問題の自然な記述は2,3行といえる．なお Floyd 流の証明は，やってみると弱い表明が17行，終了は別に2,3行計 20 行位になる．直観的な論証は，優秀なソフトウェア技術者が日常行っているところであろうが，それを記号化して相当細いレベルの証明をしても，記述の量が2倍にしかなっていないことは興味深く思われる．人間-機械系でどうやればこのレベルで証明を打切れるかは非常に本質的な問題であろう．

もう少し正確にいうと，後のほうの論証では，正確に証明を図形として書き上げたとき(いわゆる証明図)ト-記号が現れる部分は一切省略していないのである．その反面，自然数や実数についての算術レベルでの命題は全く証明しないで用いている．これらについて，いわゆる1階述語論理の証明図を書き上げようと思うと手間は飛躍的に増えるが，この部分は定理の自動証明に頼る考え方から，算術の教科書を引用しておけばよいという考え方までいろいろあり得るところである．

いいかえれば，プログラム部分の議論と，データ部分の議論が，はっきりと分かれているわけで，それは，この例題のように自然数や実数ではなく，最近かなり多くの人々が議論に参加するようになった「抽象的なデータ構造」を対象とするときに，プログラム部分での議論を変更する必要がないように，もともとの証明の体系を設計してあることを意味している [このような構成方法の原型は数学基礎論にあったものである]. なおト-論理はもっと一般的な形をしていて，ここに掲げた公理や規則がそのまま使われているわけではない．これらの多くは派生的な規則になっている．$\mathcal{F}\{A\}\vdash\mathcal{G}$ は，本来のト-論理では，1階論理を拡張した形式で

$$\mathcal{F}\supset A\vdash\mathcal{G}$$

と書かれるが，ここでつは（普通の意味の）「ならば」そのものである．Hoare の記法に近いほうが理解しやすいと思い，本稿では { } 記法を生かすことにしたのである．

4. 並行するプログラムの検証

プログラムの検証をソフトウェア工学の中心課題と

して見れば，初期にくらべて，いわゆる並行プログラムの正当性の問題が，オペレーティング・システムとの関係の深さもあって，クローズ・アップされてきたことは自然なことである．また，このように多数並行して，相互に干渉しつつ，しかも本来的には独立して，動くプログラム群は，それ自体かなり高度な知的水準をもつものとも受けとることができるかもしれない．

ここでは，E. Dijkstra が最初に論じた有名なプログラム例を取上げることにする（図-2(a)，図-2(b)）．それは，次の A,B 2つのプログラムの組であって，太枠で囲んだ部分がそれぞれのプログラムの本来的な仕事をするところで，あとは A,B 並行してうまく動作することを保証するための仕掛である．このプログラム対について論証したいことは，いくつか挙げられ

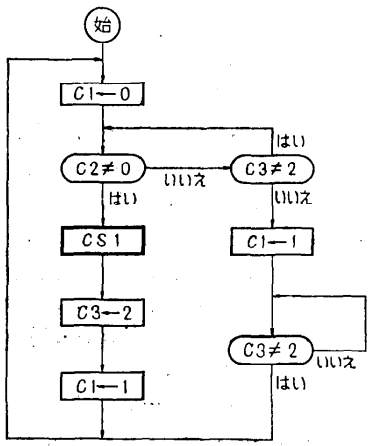

図-2(a)　並行するプログラム A

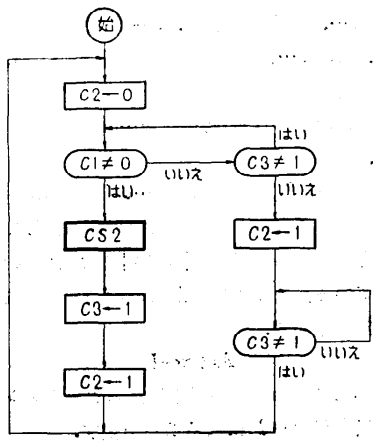

図-2(b)　並行するプログラム B

ているが，一番面白そうな問題は，十分長い時間が経過すると，$CS1$ も $CS2$ も少なくとも1回は実行されることを証明することであろう．これは「餓死問題」と呼ばれるデッドロック問題の一種である．このプログラム対の場合餓死問題が否定的に解決されることを，厳格でなく論証するとすれば，一つの論法は次のようになるであろう．

B 1. 十分長い時間が経過すると $CS1$ が少なくとも1回実行されることを証明する．それができれば，対称性から $CS2$ も少なくとも1回実行されることがいえる．

B 2. さて $CS1$ が実行されないまま（十分長く）時間が経つと，Aのプログラムのグラフ的構造［と，各のプログラムは十分長い時間が経過すれば，必ず1ステップ先に進むという，このプログラム対に関する基本的な設定と］により，Aのプログラムのコントロールは，図-3 の範囲に閉じ込められるほかない．

B 3. B2 の状態になってから図-3 のⒶ点の分岐を「はい」で通過した場合．その時刻を t_1 とする．この時 $C3 \neq 2$ である．図-3 とBのプログラムの範囲内で $C3$ を2に変える操作は含れていないから，$t_1 \leq t$ なる任意の時刻 t で $C3 \neq 2$．すなわち，この場合AのプログラムはⒶ点を含む小さいループに閉じ込められる．

B 4. B3 で述べた状態になっているとき．$C1=0$ である．なぜならば，$C1$ の値はBのプログラムによって変更されることがなく，もっぱらAのプログラムのコントロールの位置だけで定まっているからである．すなわち，点Ⓐを含む小ループに入るときは，必ず $C1=0$ となっていて，しかもこの小ループ内では $C1$ を変更することがない．こうして，$t_1 < t$ なる t で $C1=0$ & $C3 \neq 2$ となっていることとなる．

B 5. B4 の状態で時間が経つと，Bのプログラムは

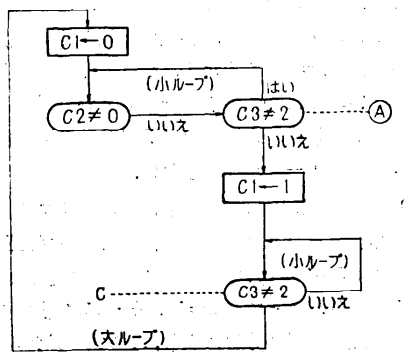

図-3　[lim1 A']

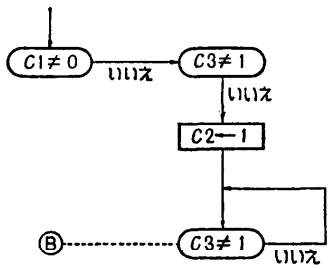

図-4 C1=0 & C3=1 のときのBの動作

図-4 の中に閉じ込められる．すなわち，Ⓑ点でループする［ただし，この際，このプログラム対の開始のとき，C3=1∨C3=2 であったと仮定する．そうすれば，以後 C3 の値が変更されるときはつねに1または2にしかならないから，この性質は保存されることになり，それゆえ，C3≠2 ならば C3=1 である．］．このとき，C2 の値は，B4 と同様の議論により，Bのプログラムのコントロールの位置のみで決定されるから，C2=1 が $t_1<t_2$ なるある時刻 t_2 以後成立ち続けることになる．

B 6. $t_2<t_3$ なる t_3 で図-2 の C2≠0 の分岐に到達したとする（B3 によりそのような t_3 が存在する）．B5 によりコントロールは「はい」の方へ進むが，このことは B3 と矛盾する．すなわち B3 のケースは生じない．

B 7. B2 の状態になってから一度もⒶ点の分岐を「はい」で通過することがない場合．B3 で述べたこと［と B5 の括弧内の仮定］により，ある時刻 t_4 から先ずっと C3=1 であるか，ずっと C3=2 であることに注目する．すなわち，Aのプログラムが図-3 に閉じ込められている限り，一度 C3=1 になったらそれは不変に保たれる．

B 8. B7 で C3=1 が保たれる場合．Aのプログラムは，B7 のケースでは，Ⓐ点の分岐を「はい」で通らないが，C3=1 であるから，それを「いいえ」でも通り得ない．またⒸ点の小ループもまわり得ない．すなわち，この場合，Aのプログラムが図-3 の範囲内で動き続けるループが存在しない．

B 9. C3=2 のとき．図-5 のようなグラフ的構造から，Bのプログラムは，いずれ同図のループに閉じ込められて C3←1 を行い，十分長い時間の後には C3=1ならしめる．よって C3=1 のケースに帰着する．

B 10. 以上の場合分けにより，Aのプログラムが図-2 の範囲内で動き続けることが不可能であるから，帰

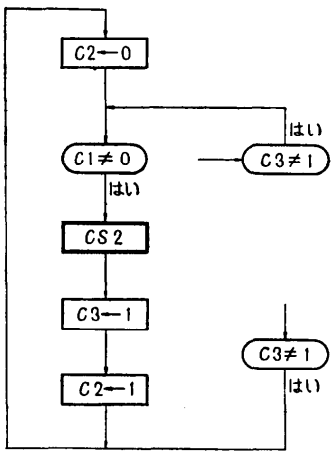

図-5 [liml(B*C=2)]

謬法により B2 が否定される． （証明終り）

さてこの厳格でない論証を記号を用いて形式的に表現することを試みることにする．

A, **B** はそれぞれ A, B の部分を表す．A′ によって，Aから $CS1$ とそれから直接出入りしている矢を取除いたプログラム部分を表す．liml **A**（あるいは liml **B**）によって，**A**（あるいは **B**）からどのループにも属さない矢や点を（記入されている計算上の指令とともに）取除いたプログラム部分を表す．x がプログラム中の変数，たとえば C1，$I=\{i_1, i_2, \cdots, i_n\}$ が自然数の有限集合のとき，$x=I$ は，

$$x=i_1 \vee x=i_2 \vee \cdots \vee x=i_n$$

の略記とする．［$x \in I$ と書いてもよいが］特に $I=\phi$（空集合）のとき，$x=\phi$ は ⅄（論理的矛盾）を表す．また，$x \leftarrow I$ は割当てステートメントの有限集合 $\{x \leftarrow i_1, x \leftarrow i_2, \cdots, x \leftarrow i_n\}$ を表す．$x \leftarrow \phi$ は ϕ と同じことになる．$I_1, I_2, \cdots, I_m$ が自然数の有限集合，$x_1, x_2, \cdots, x_m$ が変数のとき，

$$\mathbf{A} * x_1=I_1 \,\&\, x_2=I_2 \,\&\cdots\&\, x_m=I_m$$

により，

$$(x_1=I_1)\&(x_2=I_2)\&\cdots\&(x_m=I_m)$$

で表されている条件が成立つときには，決して選択されることのない分岐の矢（「はい」または「いいえ」が付いている）をすべて**A**から取除いて得られるプログラム部分を表す．

$$\mathbf{A} \Rightarrow x \leftarrow I$$

は，**A** 中にある $x \leftarrow i$ の形をしたステートメント全体の作る集合が $x \leftarrow I$ になることを表す．

$$\mathbf{A} \Leftarrow x \leftarrow I$$

は，**A**に**A**を含んでいる元のプログラムの**A**以外の部分から到達したとき必ず $x\leftarrow I$ に属するステートメントが実行され，その値が保持されていることを表す．

A: simple loop

とは，**A**が唯1個のループから成立っていることを表す（**B**についてもすべて同様）．

以上は，すべてプログラムを一べつすれば，成立っているかどうかが簡単に調べられる性質である［もしプログラムがフローチャートでなく言語によって記されていたとすれば，上記の諸性質は「シンタクス」のチェックで解る］．

さて前述の論証は，ただ1ヵ所を除くと，次の形の式だけを使いながら進めることができる．すなわち，

$$\infty x=I$$

の形か，

$$\infty\mathbf{A},\ \text{または}\ \infty\mathbf{B}$$

の形である．$\infty x=I$ は「ある時刻以後ずっと（すなわち時刻∞において）$x=I$ が成立つ」ことを表し，$\infty\mathbf{A}$ は「ある時刻以後ずっとコントロールが**A**の上にある」ことを意味する．

C1. B5 の括弧内で述べた仮定のもとで $C3=\{1,2\}$ が保証されている．この論証には別の形の式を必要とするので，ともかく結論だけをここに採用する．すると，

$$\infty C3=\{1,2\}. \tag{4.1}$$

Bはずっと動いているから，

$$\infty\mathrm{B}. \tag{4.2}$$

∞A ももちろん成立つ［これらは公理だと思ってよい］が，ここで帰謬法の仮定として，

$$\infty\mathrm{A}' \tag{4.3}$$

をおく．すると，自明な推論規則

(S1) $\infty\mathbf{A}$ ならば $\infty\mathrm{liml}\ \mathbf{A}$.

により，

$$\infty\mathrm{liml}\ \mathrm{A}'. \tag{4.4}$$

しかるに，$\mathrm{liml}\ \mathrm{A}'\Rightarrow C3\leftarrow\phi$ かつ $\mathrm{B}\Rightarrow C3\leftarrow 1$ である（図-3）．規則

(S2) $\infty x=I, \infty\mathbf{A}, \infty\mathbf{B}, \mathrm{A}\Rightarrow x\leftarrow\phi$，かつ $\mathbf{B}\Rightarrow x\leftarrow J$ ならば $\infty x=(I-J)$ または $\infty x=J$.

と，(4.1), (4.4), (4.2)から，

$$\infty C3=2\ \text{または}\ \infty C3=1. \tag{4.5}$$

C2. $$\infty C3=2 \tag{4.6}$$

と（帰謬法の）仮定をする．

(S3) $\infty x=1$ かつ $\infty\mathbf{A}$ ならば $\infty\mathbf{A}*x=I$.

により，(4.6)と(4.2)から，

$$\infty B*C3=2. \tag{4.7}$$

これと（S1）より，

$$\infty\mathrm{liml}(\mathrm{B}*C3=2). \tag{4.8}$$

しかるに，$\mathrm{liml}\ \mathrm{A}'\Rightarrow C3\leftarrow\phi$, $\mathrm{liml}(\mathrm{B}*C3=2)$: simple loop かつ $\mathrm{liml}(\mathrm{B}*C3=2)\Rightarrow C3\leftarrow 1$ であるから（図-5），規則

(S4) $\infty\mathbf{A}, \infty\mathbf{B}, \mathbf{A}\Rightarrow x\leftarrow\phi, \mathbf{B}\Rightarrow x\leftarrow J$，かつ **B**: simple loop ならば $\infty x=J$.

と(4.4), (4.8)から，

$$\infty C3=1. \tag{4.9}$$

これは(4.6)と矛盾する．すなわち(4.6)は成立たない．

C3. (4.6)が否定されたことと，もともとの(4.5)から，

$$\infty C3=1. \tag{4.10}$$

これと(4.4), (4.2)から，（S1)により，それぞれ，

$$\infty\mathrm{liml}((\mathrm{liml}\ \mathrm{A}')*C3=1), \tag{4.11}$$

$$\infty\mathrm{liml}(\mathrm{B}*C3=1). \tag{4.12}$$

以下 $\mathrm{liml}((\mathrm{liml}\ \mathrm{A}')*C3=1)$ を A'' で表す．すなわち，

$$\infty\mathrm{A}''. \tag{4.11}$$

同様に B'' で $\mathrm{liml}(\mathrm{B}*C3=1)$ を表して（図-6），

$$\infty\mathrm{B}''. \tag{4.12}$$

しかるに，$\mathrm{A}''\Leftarrow C1\leftarrow 0$, $\mathrm{A}''\Rightarrow C1\leftarrow\phi$, $\mathrm{B}\Rightarrow C1\leftarrow\phi$ である．これと，規則

(S5) $\mathbf{A}\Leftarrow x\leftarrow I$, $\mathbf{A}\Rightarrow x\leftarrow\phi$, $\mathrm{B}\Rightarrow x\leftarrow\phi$［**B**でなく全プログラムBであることに注意］かつ $\infty\mathbf{A}$ ならば $\infty x=I$.

および(4.11)から，

$$\infty C1=0. \tag{4.13}$$

これと(4.12)と(S3)から，

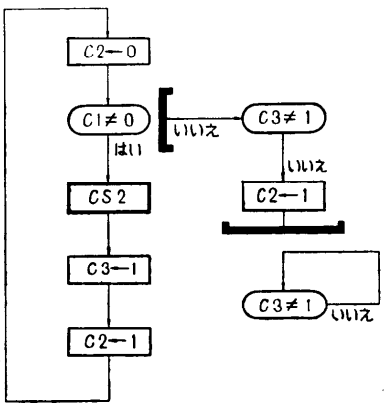

図-6 [$\mathrm{B}''=\mathrm{liml}(B*C3=1)$]

$$\infty B''*C1=0. \tag{4.14}$$

これと(S1)により,

$$\infty(\mathrm{liml}(B''*C1=0)). \tag{4.15}$$

ここで, liml(B″∗C1=0) は⑧点を通る小ループである. それゆえ, liml(B″∗C1=0)⇐C2←1 かつ liml(B″∗C1=0)⇒C2←φ. これと A⇒C2←φ について, (S5) [と A, B に関して対称な規則] を適用すれば, (4.15) より,

$$\infty C2=1. \tag{4.16}$$

C4. (4.16) と (4.11) と (S3) から,

$$\infty A''*C2=1. \tag{4.17}$$

これと (S1) から,

$$\infty\mathrm{liml}(A''*C2=1). \tag{4.18}$$

ところで, liml(A″∗C2=1) は空のプログラム (Λ で表す) である. すなわち,

$$\infty\Lambda. \tag{4.19}$$

自明な公理として,

(Ax. 1) ∞Λ でない.

を採用すれば, (4.19) と矛盾する. よって唯一の仮定である (4.3) が否定される. (証明終り)

上の証明では, いわば天下りに採用した (4.1) を証明するには次のようにすればよい. まず上述の体系の式の範囲を拡張して, ω までの順序数 (要するに $\{0, 1, 2, \cdots, \infty\}$) の部分集合 T を, 上ではもっぱら∞としたところに書いてよいものとする. そして $T\mathbf{A}$ は, 時刻 $t\in T$ なるかぎり, A のコントロールは **A** 上にあること, $Tx=I$ は, 同様 $t\in T$ なるかぎり $x=I$ が成立っていることを表すことにする. とくに $\{t\}$ は単に t と記す. たとえば, 次の規則

(S6) $t_0x=I$, $[t_0, t_1]\mathbf{A}$, $[t_0, t_1]\mathbf{B}$, $\mathbf{A}\Rightarrow x\leftarrow J$, かつ $\mathbf{B}\Rightarrow x\leftarrow K$ ならば, $t_1x=I\cup J\cup K$. ($[t_0, t_1]$ は $\{t\mid t_0\leq t\leq t_1\}$)

と, (公理として)

$$0C3=\{1, 2\} \tag{4.20}$$

を用いればよい [このような拡張は, 並行プログラムに関するより一般的な問題を扱う際に役立つと思われるごく自然な拡張として言及したのであるが, (4.1) を「論証」するためのもっと手っ取り早い方法は, A, B の「始」の直前に $C3\leftarrow\{1, 2\}$ がついた (各2本の) 矢が書いてあったと思うことである. すると, $A\Leftarrow C3\leftarrow\{1, 2\}$, $B\Leftarrow C3\leftarrow\{1, 2\}$. 一方 $A\Rightarrow C3\leftarrow\{1, 2\}$, $B\Rightarrow C3\leftarrow\{1, 2\}$ であるから, 次の自明な規則

(S7) $\mathbf{A}\Leftarrow x\leftarrow I$, $\mathbf{B}\Leftarrow x\leftarrow J$, $\mathbf{A}\Rightarrow x\leftarrow K$, $\mathbf{B}\Rightarrow x\leftarrow L$, ∞**A** かつ ∞**B** ならば $\infty x=I\cup J\cup L\cup K$.

と (公理) ∞**A**, ∞**B** から (4.1) が得られる. これならば式を拡張しないで済む].

以上の証明は, シンタクティックないし図的なチェックを別とすれば約 20 ステップに過ぎない. 元の厳格でない証明は, 多少まわりくどくなっているが, 40 行を越えている. これで見ると適当に記号化された証明が, 厳格でない証明よりも短くおさまる可能性があるといえる [最初の論証は, 後の証明のまねをすれば多少短くなる (興味のある読者は試みていただきたい). 多くの人々は最初の論証があるから記号化が短くできたと思われるに違いないが, 少なくとも今の場合, 実情は逆であって, 記号化の過程で必要な概念が整理され, さらに証明図 (証明全体の木構造をしたパノラマ) を書くことによって, より短い証明ができたのである]. 論証であるか, 証明であるかは別として, これらをプログラム自体と同様人と人との間で, ひいては人と機械, さらには機械と機械との間でコミュニケートしなければならなくなることはソフトウェア工学のすう勢といえようが, そこには人工知能学的な多くの深い問題が含まれている*.

5. 終 り に

ト-記号を用いた体系は, より一般的な形で 1974 年シュツットガルトでの IFIP WG 2.2 で提出したものである. これは Séminaire IRIA 1974 (M. Nivat ed.) に収録されてもいるが, 日本国内からは事実上アクセスできないと思われるので, 機会を改めて詳しく述べることにしたい**. 理論はともかくとして, 「論証」が少しずつ記号化されて行く段階を見ていただければ, 本稿の主目的は果たされたと思う. また, この場合, 記号化された論理が即いわゆる既成の数理論理ではないことも気づかれたに違いない. 検証にかぎらず, 何を本質的な概念としてとらえ, 形式化または記号化するかは, IFIP に関係した共同研究が活発に行なわれている分野の一つである.

(昭和53年8月23日受付)

---

* 拡張した形式化は非常に強力なので並行プログラムについて従来から扱われてきた典型的問題群はほぼすべて取扱えるであろう. 本稿ではシクエンシャルなプログラムに対する方法が最も無力になっている「餓死問題」が, ただ1つの「時制」記号∞だけで, 厳格に表現できかつ証明できることを示すことに主眼を置いた. 「時制」論理 (tense logic) は, 最近のプログラム理論, 人工知能論の1つのテーマでもある.

** たとえば, (R8) $\mathcal{P}\&C\{A\}\tau\mathcal{Q}$ と $\mathcal{P}\&C\{B\}\tau\mathcal{Q}$ から $\mathcal{P}\{\textit{if } C \textit{ then } A \textit{ else } B\}\tau\mathcal{Q}$ を規則 (R1)～(R7) に付加すれば, プロシージャーのないプログラムの論証には十分である. 全体の体系は PASCAL 型のパラメータ・メカニズムを持つ (一般的にはリカーシブな) プロシージャーについて作られている.