6. プログラムの理論について

小野寛晰（津田塾大学）

1. はじめに

プログラムの理論についての概観をのべる。プログラムの理論は普通MTC（methematical theory of computation）とか理論プログラミング（theoretical programming）という名でよばれている。ここではプログラムの理論が何を目標とし，どのような限界を持ちまたどのような発展の可能性があるかについてのべてみたい。話を具体的にするために，inductive assertion methodとよばれる方法を中心にして話を進める。

プログラムの理論は大ざっぱな言い方をすればプログラムという形をしたアルゴリズムについての研究であるといえる。特にプログラムの形式とプログラムの意味する内容との相互関係を明らかにしていくことに重点がおかれる。具体例として，あたえられたプログラムの正しさを示すという問題をとりあげてみる。プログラムを書いた時におこりうる誤りとしては文法的なミスなどのようにいわば表現上の誤りと，アルゴリズムとしての誤りとが考えられる。後者のタイプの誤りを検出するために通常テストデータによるデバッグという方法がおこなわれている。この方法による場合には，いくつかのテストをパスすれば誤りがとり除かれたと考えられこのプログラムは正しいと結論される。しかしこのような意味で「正しい」プログラムを使って得られた結果は十分に信頼するにたるものであるといえるだろうか。たとえば，計算機を用いて得られた数学上の結果について，数学の定理と同程度にその正しさを主張しようとするには，定理に対する証明と同じように，その結果を得るのに用いたプログラムが正しいことを何らかの形で示す必要があるのではないだろうか。つまりテストデータによる方法が十分信頼するにたるものであるというならば何故この方法でよいのかを示し，またそうでないならばそれにかわる方法によりプログラムが正しいことを示す必要があると思われる。現在のところ，テストデータによる方法が十分信頼にたるものであることを保証する結果は何も得られていない。他方，プログラムの正しさを証明する方法として，これからのべるinductive assertion methodなどいくつかの方法が知られている。勿論それらの方法で証明をいかにおこなうかを考えてみると，いくつかの問題点がみいだされる。とくにこの方法を実際に適用する際に現われる困難とわずらわしさを無視することはできない。だからといってプログラムの正しさを確かめるためのより合理的な方法を追求する必要性がうすれるわけではない。

プログラムの理論における問題としては上述の正しいか（正当性）という問題のほかに，プログラムがとまるか（停止性）とか，二つのプログラムが同じ機能を持っているか（同値性）といったものをあげることができる。こういったプログラムの諸性質の概念をはっきりと定義し，その上であたえられたプログラムがそれらの性質をもつかどうかを示す具体的な方法を提示することが必要である。

2. Inductive assertion method

Floyd，Naur等によるinductive assertion methodについてのべてみる。この方法はプログラムの正しさをそのプログラムを構成している各要素の正しさに帰着しようとするものである。まずプログラムが正しいとはどういうことかを考えておこう。プログラマはある意図をもってプログラムを書く。作られたプログラムはそれ自身で一つの意味を持つ。アルゴリズムとしての誤りというのは，プログラマの意図とプログラムの意味

のずれとしてとらえることができる。すなわち，「あたえられた意図について」プログラムが正しいとは，その意図とプログラムの意味とが一致していることに他ならない。したがってプログラムが正しいという場合にはプログラマの意図といったものがあらかじめあたえられていなければならない。そのためには，その意図を何らかの形で表現する必要がある。たとえば図1のプログラムに対する意図を「aをbで割った時の商がy，剰余がxである」とした時，それをつぎのように表わしてみることにする。

「入力a，bについて $a \geq 0$，$b > 0$ がなりたつならば，出力x，yは $a = b \times y + x$ および $0 \leq x < b$ をみたす」。

さて図1のプログラムがこのような意図に関し正しいものであることを示してみる。まず番号のつけられた各点でx，y，a，bの間にどのような関係がなりたつかを予想し，それを論理式の形で表現しておく。以下では $\wedge$，$\vee$，$\supset$，$\neg$はそれぞれ「かつ」，「または」，「ならば」，「…ではない」ということを表わす。

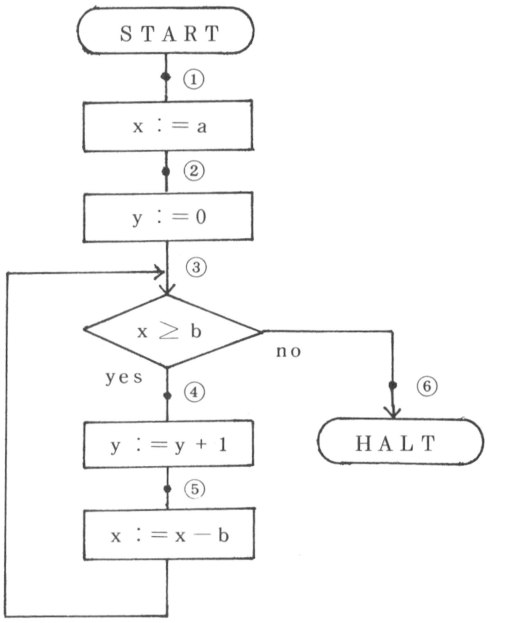


図　1

上の意図により，①では

A 1　　$a \geq 0 \wedge b > 0$

がなりたち，⑥では

A 6　　$a = b \times y + x \wedge 0 \leq x < b$

がなりたたたねばならない。入力a，bが①をみたしている時，他の各点ではそれぞれつぎのような関係がなりたつと予想しておく

A 2　　$x = a \wedge x \geq 0 \wedge b > 0$　　……②

A 3　　$a = b \times y + x \wedge x \geq 0 \wedge b > 0$　　……③

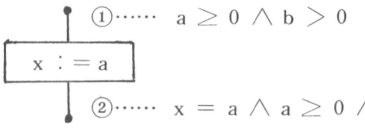
A 4　　$a = b \times y + x \wedge x \geq b \wedge b > 0$　　……④

A 5　　$a = b \times (y - 1) + x \wedge x \geq b \wedge b > 0$　　……⑤

ただし，これらの式においてx，yはその番号の計算がおこなわれる際のx，yの値を表わすものとする。

A 1からA 6までの論理式では inductive assertion とよばれる。さてプログラム全体の正しさはつぎに示されるように，各構成要素の正しさのつみ重ねの結果としてえられる。

1）①から②へ計算がおこなわれる時



②で命題P（x，y）がなりたっていると仮定する。x ：＝aによりxの値がaになりそのxについてP（x，y）がなりたつのだから，　x ：＝a　という代入文が実行される前（つまり①）にはP（a，y）がなりたつはずである。ところで①では　$a \geq 0 \wedge b > 0$　がなりたっているのだから結局

（$a \geq 0 \wedge b > 0$）$\supset$ P（a，y）

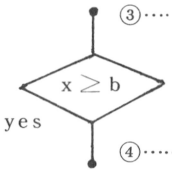
がなりたっていればよいことになる。ところでいまの場合P（x，y）は　$x = a \wedge a \geq 0 \wedge b > 0$　だから

P（a，y）は　$a = a \wedge a \geq 0 \wedge b > 0$，したがって

V 1.　$(a \geq 0 \wedge b > 0) \supset (a = a \wedge a \geq 0 \wedge b > 0)$

がなりたっていればよい。ところであきらかにV 1は正しい式だから，結局①でA 1がなりたつならば②でA 2がなりたつことがわかった。同様にして，②でA 2がなりたつなら，③でA 3がなりたつことを示すことができる。

2）③から④へ計算がおこなわれる時



③……　$a = b \times y + x \wedge x \geq 0 \wedge b > 0$

④……　$a = b \times y + x \wedge x \geq b \wedge b > 0$

④で命題Q（x，y）がなりたつとする。$x \geq b$ という条件で③から④へ計算がすすむのだから③では $x \geq b \supset Q(x, y)$ がなりたたなければならない。したがって

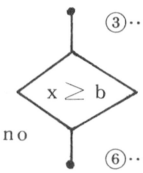
$(a = b \times y + x \wedge x \geq 0 \wedge b > 0) \supset (x \geq b \supset Q(x, y))$，

すなわち

V 2.　$(a = b \times y + x \wedge x \geq 0 \wedge b > 0) \supset (x \geq b \supset (a = b \times y + x \wedge x \geq b \wedge b > 0))$

がなりたつことを示せばよい。ところでこれは正しい式になっているから③から④へ計算がすすんだ場合，③でA 3がなりたてば④でA 4がなりたつことがわかる。④から⑤，⑤から③へ計算がおこなわれる場合は 1）の場合と同様にして確かめられる。

3）③から⑥へ計算がおこなわれる時



③……　$a = b \times y + x \wedge x \geq 0 \wedge b > 0$

⑥……　$a = b \times y + x \wedge 0 \leq x < b$

2）の場合と同様におこなうと

V 3.　$(a = b \times y + x \wedge x \geq 0 \wedge b > 0) \supset (\neg(x \geq b) \supset (a = b \times y + x \wedge 0 \leq x < b))$

がなりたつことを示せばよいことがわかる。ところでV 3は確かに正しい式である。従って③でA 3がなりたてば⑥でA 6がなりたつことがわかる。

以上の証明をつなぎ合わせると，①で　$a \geq 0 \wedge b > 0$　がなりたつならば⑥で　$a = b \times y + x \wedge 0 \leq x < b$　がなりたつことが示されたことになる。ここで注意が必要なのは上でおこなったのは厳密には「計算が終了するならばこのプログラムは正しい」ということの証明にすぎないということである。というのは，③ → ④ → ⑤ → ③とまわるループをいつかは抜け出して計算が③から⑥へおこなわれるということはなにも保証されていないからである。したがって「計算がいつかは終了する」ことをあらためて証明する必要がある。上の証明の中で inductive assertion から作られたV 1，V 2，V 3等の式は verification condition とよばれている。

上でのべた証明法を整理して一般的な形でのべておこう。プログラムPがフローチャートの形であたえられているとする。またPに対する意図が「$\varphi(x_1, \cdots, x_m)$をみたすような入力$(x_1, \cdots, x_m)$に対するPの出力が$(y_1, \cdots, y_n)$ならば$\Psi(x_1, \cdots, x_m, y_1, \cdots, y_n)$がなりたつ」という形であたえられているとする。まずPの中にあらわれる各「矢印」に番号①，②，…，ⓝをつけておく。とくにSTARTから出る矢印には①を，HALTに入る矢印にはⓝをつけておくことにする。図1のプログラムの証明では各番号にinductive assertionを対応させたが，必ずしもそれは必要ではない。本質的なのは各ループのどこかにinductive assertionをつけておくことである。そこで，このことを考慮に入れつぎの1)から5)までをおこなう。

1) つぎのa，bの条件をみたすような番号の集合C($\subset \{①, ②, \cdots, ⓝ\}$)を一つ定める。

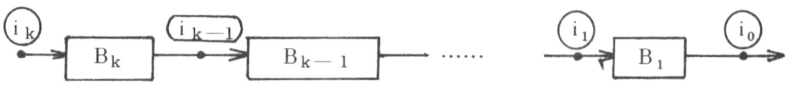
a．①，ⓝ $\in$ C

b．Pの中の各ループはCの要素で番号づけられているような「矢印」を少くとも一つは含む。

2) Pの中の道でその両端はCに属すが，両端以外の番号はCに属していないようなもの全体の集合Sをとる。Sはあきらかに有限集合である。

3) Cの各番号にinductive assertionを対応させる。特に①に対してはφをⓝに対してはΨを対応させる。

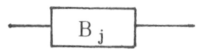
4) Sに属する道αに対し，verification condition $F\alpha$をつぎのようにしてつくる。いまαが

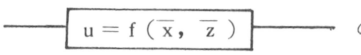


の形であるとしておく。また(i_0)，(i_k)にはそれぞれinductive assertion $A(x_1, \cdots, x_m, z_1, \cdots, z_n)$，$A'(x_1, \cdots, x_m, z_1, \cdots, z_n)$があたえられているとする。まず論理式$Q_j(x_1, \cdots, x_m, z_1, \cdots, z_n)$ $(j = 0, 1, \cdots, k)$をつぎのように帰納的に定義する。(以下$Q_j(x_1, \cdots, x_m, z_1, \cdots, z_n)$を単に$Q_j(\bar{x}, \bar{z})$のように略す。)

$$Q_0(\bar{x}, \bar{z}) \equiv A(\bar{x}, \bar{z})。$$

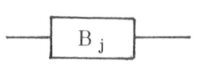
$Q_{j-1}(\bar{x}, \bar{z})$がすでに定義されているとする。

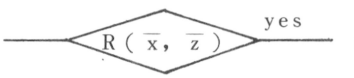
i) 

が 

の形のとき。

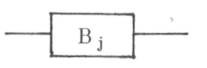
$Q_{j-1}(\bar{x}, \bar{z})$の中の変数uをすべて$f(\bar{x}, \bar{z})$でおきかえた式を$Q_j(\bar{x}, \bar{z})$とする。

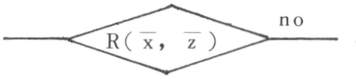
ii) 

が 

の形のとき。

$$Q_j(\bar{x}, \bar{z}) \equiv R(\bar{x}, \bar{z}) \supset Q_{j-1}(\bar{x}, \bar{z})。$$

iii) 

が 

の形のとき。

$$Q_j(\bar{x}, \bar{z}) \equiv \neg R(\bar{x}, \bar{z}) \supset Q_{j-1}(\bar{x}, \bar{z})。$$

$Q_j(\bar{x}, \bar{z})$の作り方からあきらかなように，計算が(i_k)から道αをとおって(i_0)に到った時に(i_0)で$A(\bar{x}, \bar{z})$がなりたつならば(i_k)では$Q_k(\bar{x}, \bar{z})$がなりたっていなければならないことがわかる。したがって

$$F\alpha(\bar{x}, \bar{z}) \equiv A'(\bar{x}, \bar{z}) \supset Q_k(\bar{x}, \bar{z})$$

が (i_k) でなりたっていればよい。この F_α を道αに対する verification condition という。

5) Sの各要素に対する verification condition が正しいことを示す。

この 1) から 5) までがおこなわれた時にプログラムが正しいことの証明が完了する。

原理的には，このような方法でプログラムの正当性の証明がおこなわれる。しかし，プログラムの形によっては上の方法にいくらかの修正を加える必要がある。たとえば配列を含むような場合には，上のままではうまくいかないことがある。これは a〔i〕という表現が，配列名 a と添字の i という二つの変数からなっていることに原因がある。

3. 停止性の証明

上でおこなった正当性の証明は，「停止するならば正しい」ことを示したにすぎなかった。そこで正当性の証明を完結させるために，プログラムが停止することの証明をどのようにおこなうかを図1のプログラムについてのべることにする。前にのべた証明から①で $a \geq 0 \wedge b > 0$ がなりたっているならば③では $x \geq 0 \wedge b > 0$ がなりたつことがわかる。ところで①から③に到り，さらにループを i 回まわって③にきたときの x の値を x_i とすると

$$x_0 = a$$

$$x_{i+1} = x_i - b \quad \text{ただし} \quad x_i \geq b \quad (i \geq 0)$$

となる。したがって $x_i \geq b$ である限り，$b > 0$ および $x \geq 0$ という条件から

$$x_0 > x_1 > x_2 > \cdots\cdots \geq 0$$

という列がえられる。もし計算が終了しないなら，いつまでもループをまわりつづけるから注意の n に対し x_n が定義されることになる。ところでこれは自然数の無限下降列が存在することになり矛盾がおきる。したがっていつかは③から⑥の方へ計算が進み停止しなければならないことがわかる。

このようにプログラムが停止することの証明はプログラム中のある変数ないし変数の組に着目し，その値がある順序のもとで減少するが，無限に減少しつづけることはないということを示すのが一般的なやり方である。

4. 問題点

以上にのべたように inductive assertion method はプログラムの正しさを証明する一つの方法となっていることがわかった。そしてこの方法によりプログラムの正しさが証明できたならば，数学の証明と同程度に人を納得させることができるといえよう。しかしながらこれで問題はすべて片づいたわけではない。ここで inductive assertion method がかかえている問題点を検討してみる必要がある。

4.1 理論上の問題点

図1のプログラムの例からもわかるように inductive assertion method のキイ・ポイントは各点につける inductive assertion の選び方にある。プログラムが複雑になれば当然 inductive assertion も複雑なものになり，どのような論理式を選んだらよいのかが難しくなる。そして inductive assertion の選び方がまずければ，プログラムが正しいにもかかわらず正当性の証明ができないということもおこりうる。それでは，プログラムが正しい場合には必ず inductive assertion を選ぶことができるかという疑問が生じてくる。これに対してはつぎのような答がえられる。「集合論的な意味では inductive assertion は存在する。しかしながらそのような inductive assertion が論理式として表現できるか否かは，論理式の表

現に用いることが許される言語の選び方に依存する」。たとえばプログラムの形だけから（つまりセマンティクスに依存しないで）あたえられた点における inductive assertion を述語論理の論理式を用いて表わすことは必ずしもできないということが知られている。表現に使用する言語の選び方はつぎのような問題とも関連している。それはプログラムの正しさの証明をする場合，つぎの二通りの必要性が考えられるということである。

1) プログラムを書いた人がプログラムに虫のないことを知るため。

2) サブルーチンのプログラムのように何らかの目的のために他人にプログラムの正しさを説明するため。

前者の場合には証明にどのような言語を使ってもよいし，自分で納得できるものである限りどんな立場で証明をおこなってもよい。しかし後者の場合には共通な理解を得るために限定された言語を用い限定された範囲での証明をすることが望ましい。そのためにはたとえば証明そのものを形式化するということが考えられる。つぎに述べるように正当性の証明を計算機におこなわせようとする場合には，この証明の形式化ということは不可避になってくる。

4.2 実用化の上での問題点

実用上の問題点は，証明が煩雑であるという一言につきると思われる。すでにのべたように inductive assertion を選ぶ一般的な方法はある意味において存在しないから，inductive assertion はうまく選ぶ必要がある。さらにうまく選ぶことができたとしても一般にはそれは非常に長い式となり，その結果として，verification condition をつくりその正しさを証明するという過程で誤りをおかしかねないというディレンマが生ずる。そこでこれらの煩雑さからのがれるために正しさの証明を計算機におこなわせるということが考えられる。とりあえず，すでに述べた正当性の証明の各ステップを計算機がおこなうことが可能かどうかを検討してみよう。

1)，2) の集合Cと集合Sをとることは可能である。3) の inductive assertion を対応させるのはすでにのべたことから一般には不可能であろう。4) の verification condition を作る段階は可能である。この場合，つくられる verification condition をそれと同値で出来る限り簡単な論理式でおきかえておくということまでおこなっておくのがよいだろう。5) の verification condition の証明については二通りの考え方がある。第一のものは theorem prover を使うという考えである。threorem prover の能力についてはいろいろ評価のしかたがあろうが，その能力に決定的な限界があるというのは一つの事実である。第二の考え方は計算機は論理式の簡単化とか，えられた証明のチェックのみに用い，発見的な要素が必要である証明そのものは人間がおこなうというものである。実際に計算機を使って正当性の証明をおこなわせる試みがいくつかなされているが，十分成功したとはいい難いようである。

5. その他の話題

inductive assertion method はプログラムの各構成要素の意味づけから出発していた。これに対しプログラムをグローバルに眺め，プログラムの意味をそのプログラムが定める入出力間の関係，すなわちひとつの関数としてとらえることもできる。プログラムの諸性質をその立場に基いて証明するという方向のうち代表的なものが Scott の研究である。

たとえばつぎのような recursive program において，fはいくつかの解を持つ。

f (x) = if x = 0 then 1 else x × f (x − 1)。

たとえば

$f_1(x)$ = if $x \geq 0$ then $x!$ else undefined

$f_2(x)$ = if $x \geq 0$ then $x!$ else 0

などが解となっている。Scott は，上のプログラムの「意味」はこれらの解のうち「最小」のもの（最小不動点）であると考え，最小の解についての性質を証明する一つの方法をあたえた。

プログラムの諸性質を証明するという方向では，Scott の方法と inductive assertion method が代表的なものである。この方向の応用としてはコンパイラとかオペレーティングシステムの正当性の証明があげられる。またいままではプログラムの正当性といっても，非数値的なプログラムをとりあつかうことが多かったが，数値計算のプログラムの正当性の証明というのも興味ある問題であろう。この場合，正しいとは何かということとを今一度検討しなおしてみる必要がある。たとえば具体的にプログラムの形としてあたえられた数値計算のアルゴリズムに対する誤差解析と正当性の証明法の間には密接な関係があると思われる。

作ったプログラムに対しその正当性を証明するという方向に対し，むしろ正しいプログラムを作る方法を考えていくという方向もある。勿論，これは非常に難しい問題点をかかえている。現状は，具体的な方法を模索している段階であろう。

最後に，以上の話に関係した文献をいくつかあげておく。

〔1〕 R. W. Floyd, Assigning meaning to programs, Proc. of a symposium in applied mathematices, 19 AMS (1967) 19 — 32.

〔2〕 Z. Manna, Introduction to mathematical theory of computation (1974) (McGraw-Hill).

〔3〕 Z. Manna, S. Ness and J. Vuillemin, Inductive methods for proving properties of programs, Proc. of an ACM conference on proving assertions about programs (1972) 27 — 50.

〔4〕 J. McCarthy, A basis for a mathmatical theory of computation, Computer programming and formal systems (1963) 33 — 70 (North-Holland).

〔5〕 小野寛晰，プログラムの基礎理論，津田塾大学 数学セミナー・ノート No. 2 (1974).