組み込みソフトへの数理的アプローチ

第2回 プログラムの証明 ── ホーア論理とダイクストラのプログラム検証法

藤倉 俊幸

前回は，設計が要求に対して正しいことを証明することを扱った．設計が正しくできたら，次は正しく実装（プログラム）しなければならない．今回は設計から実装に至る部分について説明する．

プログラムが正しいことを証明しようとするアプローチは1960年代後半ごろから研究されている．けっこう歴史があるわりには利用されていない．実際に自分のプログラムが正しいことを証明したことがある人はいるのだろうか．テストはするけれども，証明は恐らくやらないのではないだろうか．

1 ピタゴラスの定理の場合

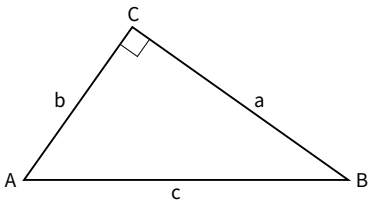
● 計算さえできれば良いのか

証明といえば数学の世界の話で，プログラミングとは関係ないと思うかもしれない．しかし，その数学も昔は計算ができればよいという時代もあった．

ピタゴラスの定理（**図1**）は，直角三角形の辺の長さに関する定理である．この定理の証明方法は，ギネスブックによると500以上あるらしい(1)．真であると証明された命題を定理と呼ぶが，ピタゴラスが証明する前から直角三角形のこの性質は知られていて利用されていた．

たとえば，辺の長さが3：4：5ならば直角三角形であることは，エジプト人や縄文時代人も実は知っていて直角を作り出すのに利用していた．直角を作り出せなければ，クフ王のピミッドも三内丸山遺跡の大型掘立柱建物も作れない．

ただし当時は，証明などというものは存在しなかった．実用的な直角を作り出す手順として利用されていただけで，実際に作ったものが倒れてしまうとか，何となく傾いているとか，そういうことがなければ良かったのかもしれない．この状況は，現在のソフトウェア開発と似ている．



直角三角形ABCで，
$a^2+b^2=c^2$
が成り立つ．
逆に，上式が成り立つような3辺a，b，cをもつ三角形は直角三角形である

図1 ピタゴラスの定理

● なぜ証明が必要になるのか

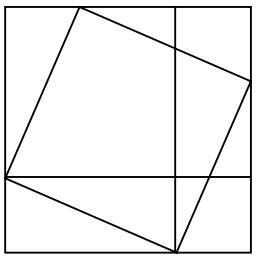
ピタゴラスの定理の記述と，「3：4：5ならば直角三角形になる」ということは，適用可能な範囲がまったく違う．記述の抽象度が上がってくると，テストはできなくなるので証明が必要になるのだ．

3：4：5のほうであれば，ひもなどを使ってすぐに実験することができる．これをテストと呼ぶのは問題かもしれないが，3：4：5のところに結び目でもつけてピンと張れば直角が出現することを実際に目で見ることができる．

一方，ピタゴラスの定理のほうは，このような目に見える実験が困難なので，どうやって納得したら良いのか見当が付かない．説得力のある説明が欲しくなる．納得するために，いくつも直角三角形を描いて辺の長さを測ったり，$a^2+b^2=c^2$を満たす棒を三本用意して三角形を作ると直角になることを試してみるのも方法であるが，実験には誤差がつきもので，必ずしも定理どおりの結果が得られるとは限らない．

実測値が期待どおりにならないとき，実験のほうが失敗していると主張するのが定理なのである．定理というものはそれほど強力なものだから，その強力さを実験で納得することは難しい．もし，定理のように強力なソフトウェア部品をそろえることができれば，ソフトウェアの品質を高めることができる．そのために証明が必要になる．

ヒンドゥー式の証明は，**図2**のような図を描いて，図の横に「見てのとおり」と書いてあるだけらしい(2)．これを見て納得す



見てのとおり

図2 ヒンドゥー式証明

KEYWORD ── プログラムの証明，ホーア論理，ダイクストラ検証法，形式手法

ホーア

▶{A}P{B}
- Aのとき，Pを実行すると，Bになる

▶表明付きプログラムを利用する
- まずプログラムが必要

▶公理・推論規則を使って証明する
- 代入文の公理
- while文の公理などがある

▶例
- $\{x>=0\}x=x+1\{x>=1\}$

ダイクストラ

▶A→wp(P，B)
- Aのとき，Pを実行すると，Bになる

▶述語変換子wp(P，B)
- BになるようにPを作る

▶等式を証明する
- A=wp(P，B)
- 方程式を解くようにプログラムを作る

▶例
- $(x>=0)=wp(x=x+1, x>=1)$

P：プログラム　A：前条件　B：後条件　wp：最弱前条件

図3　ホーアとダイクストラ

るのには，ある程度のセンスが必要である．ある種の特殊なセンスのある人にだけ説得力がある点ではUMLと似ている．しかし，説得力はあるが，証明とはいえない．証明と言うためには，万人に開かれたもう少していねいな心遣いを期待したい．

2 プログラムの証明

● プログラムが先か，後か ——ホーア論理とダイクストラ検証法

プログラムの証明は，フローチャートの検証をフロイド(Floyd)が始めて，それをホーア(Hoare)が改良してホーア論理にまとめたのが始まりだといわれている(**図3**)．フロイドの前にチューリング(Turing)が表明(assertion)という形式を提案していたという話もある．ホーア論理に続くダイクストラ(Dijkstra)のプログラム検証法は，プログラムを抽象的なレベルから検証しながら開発するスタイルの原型ではないかと思う．

ホーア論理を使用した表明付きプログラムの証明や，ダイクストラと関連する最弱前条件については，「組み込みプログラミングノウハウ入門」の中でも解説したことがある[注1]．このときは，ホーア論理だけでは組み込みシステムには使えないという話をした．組み込みシステムで重要な並列性などを考えると確かに使用できないが，一つの関数を考えてそれが正しく作られていることを証明するような場合には使用できる．

また，VDMやZなど[注2]で作成した形式仕様記述をプログラムに反映させる際の枠組みとしても利用できる．

歴史的には，ホーア論理が先にあって，それからダイクストラの検証法が生まれた．ホーア論理は，プログラムと仕様がまず与えられて，そこからプログラムの正しさを証明する手法である．

一方，ダイクストラは方程式を解くように，仕様からプログラムを作ることを目的としていて，プログラムは与えられるのではなく，最弱前条件の規則を使って仕様からプログラムをトップダウンに検証しながら作っていくスタイルになっている(開発検証と呼ばれる)．この点ではダイクストラのほうがおもしろいし，実用的である．ちなみに，ダイクストラはセマフォの生みの親としても有名だ．

● 配列の中で最大の要素を探し出す例

例として，ある整数を要素とする配列の中で，最大の要素を探し出すプログラムX0を考えてみる．

まず，「整数を要素とする配列Aの中で最大の要素」を形式的に表現する必要がある．形式的というのは数式か論理式でと

注1：本誌2003年1月号，組み込みプログラミングノウハウ入門 第9回
注2：VDM(Vienna Development Method)はモデル・ベースの形式手法．Zは仕様記述言語の一つ．

Column

形式手法とは

形式的というとおもに二つの意味がある．たとえば，形式主義を辞書で調べると，

けいしきしゅぎ【形式主義】(formalism)
A）一般に事物の形式を重んずる結果，内容そのものを軽視または無視する態度．
B）科学理論を，対象の形の面に認められる性質や形に関する規則にもとづいて構成する立場．

の二つが見つかる．ここで意識しているのはBの立場でソフトウェアを開発したいということである．次に，formalismを辞書で調べると，

formalism【名】
(科学的論証の)数学[論理]的形式；〔数学〕形式主義； cf. logicism.

と説明され，さらにlogicismを調べると，

logicism【名】〔哲学〕論理主義，論理に依存して事物をとらえようとする立場．

となる．つまり，ソフトウェア開発で形式手法とか形式的手法といった場合は，勘や経験だけではなく数学的あるいは論理的な方法でソフトウェアを作るための手法という意味になる．ある形式手法の研究会で，形式手法を広めるために名前を変えたほうが良いのではないかという話題があった．そのとき，数理的手法にしたらどうですかと提案したが，それでは範囲が広すぎると却下されてしまった．現在，形式手法というと，モデル検査，定理証明，形式仕様記述のことを一般的には指している．だから，数学応用開発手法のような名前では現状とのギャップがありすぎる．ちなみに，この連載は，使えるものは何でも使うので範囲を広げすぎた名前にした．

数学の論理の中に，形式論理というものがある．普通の論理と形式論理でどこが違うのか．形式論理は推論を「形式化」して記号操作にしてしまう体系である．内容を捨て去った形式だけで推論を行う論理である．このときの「形式」は上記Aの意味に近くなる．最終的には，コンピュータ・プログラムのように記号操作だけでソフトウェアを開発できるようになるかもしれない．しかし，ソフトウェア開発の現状は数学より2000年以上遅れていて，そろそろ証明ぐらいしてみようかという段階である．