

# プログラムの正当性証明入門

## 2

シドニー・L・ハントラー ジェイムス・C・キング

訳 玉井哲雄

### 4. 無限記号実行木と帰納法

プログラムは有限個の文からなる。無限記号実行木の節点はプログラムの文でラベルづけされているから、文のラベルの中には、無限回現われるものがなければならない。したがって、記号実行木の無限をなす部分は、プログラムの繰返しから生成されることになる。われわれのプログラミング言語の場合は、繰返し構造をなすものは DO WHILE 文のみである。

各繰返しをたどる路は、“カット”（しるし）をそれぞれの繰返しに少なくとも1つおくことにより、分離される。

これから述べる帰納法は、繰返しが1カ所以上でカットされても成り立つから、プログラムの2つの文のあいだすべてにカットを入れて、繰返しをみな切断することも、自明なやり方ながら可能である。一般には、各繰返しを一度だけカットする方法がとられる。われわれの言語では、カットは各 DO WHILE 文とその繰返しの本体とのあいだにしるしをおくことによって作られる。

以下の議論では簡単のために、PROCEDURE 文の直後にもカットが置かれているものとしよう。そして、カットの一つから始まるプログラムの記号実行を想定してみよう。

このようにプログラムの任意の点から始めた場合、“入力”は実際はプログラムの状態で表わされるので、すべ

てのプログラム変数をそれぞれ一つの記号値で初期化し、pc は真に初期化すればよい。この点からの記号実行は、プログラム変数の値に依らず、実行がこのカットに至るようなあらゆるケースを代表する。すなわち、新たに設定された記号値が、あらゆるケースを表わすことになる。

記号実行は、その後のカットがプログラムの最後の RETURN に達すれば止まる。プログラムの各繰返しはカットされているので、この記号実行はどんな場合についても停止し、有限の記号実行木をもつことになる。

カットから始まるこのような記号実行をそれぞれカット（記号）実行と呼び、それに対応する木をカット（記号実行）木と呼ぶ。もしカット C に ASSUME 文をおき、その C のカット木の終りとなるカットに PROVE 文をおけば（RETURN 文にはすでにその直前に PROVE 文がおかれている）、C のカット実行に対する正当性の証明（これらの入出力表明に関して）を論ずることができる。

カット木が有限であるので、前節で述べたような証明を常に試みることができる。もしこの証明がうまくゆけば、カットの入力表明を満たす値によってこのカットから始められるあらゆる実行は、別のカットに至ることが保証され、その点でのプログラムの値は、対応する出力表明を満たす。

手続き全体の正当性証明は、カット実行の証明から構成できる。必要とされるのは、各カット木で仮定される入出力表明と、部分から全体の証明を作りあげるための、明確な論法である。各カットにある適切な表明を入れることが、この両者を可能にする。“適切な”という言葉を使ったのは、これらのカット表明（普通は、“帰

Copyright © 1976, Association for Computing Machinery, Inc.

Sidney L. Hantler and James C. King, An Introduction to Proving the Correctness of Programs, acm computing surveys, Vol. 8, No. 3. Sept. 1976.

納表明”ないし“帰納述語”と呼ばれる)が、数学的帰納法による通常の証明の帰納仮定に対応し、発見するのが大変難しいことが多いからである。

仮に、最初のカットを除く各カットに、適切な表明が ASSERT (クブール) という式形の新たな文を用いて、ある“帰納法の天才”によって、挿入されたとしよう。

図6は、図4(先月号)の GCD 手続きのカットと帰納表明を示したものである。ASSERT 文に対し、それが置かれた文脈によって、2種の記号実行の定義がなされる。カット実行を、ASSERT 文が最初の(変数にすべてそれぞれの記号が定められた直後の)文となるように、改めて考えよう。(この場合、実際のカットのしるしは、ASSERT 文の直前に置かれていると想定する。)

この文脈で ASSERT 文を実行する際は、それが ASSUME 文であるのとまったく同様に扱われる。すなわち、それはカット実行の入力表明を与える。また、カット実行を終了させるカットが出てくると、それに応じた ASSERT 文は、それが PROVE 文であるのと同様に実行される。(この場合、実際のカットのしるしは、ASSERT 文の直後に置かれていると想定する。)1つの ASSERT 文が、文脈によって ASSUME 文とも PROVE 文とも取り扱われることに注意されたい。

プログラムの初め (PROCEDURE) と終り (RETURN) という特別な場合は、明らかではあるが、ふれておかなければならない。

PROCEDURE 文の後の最初のカットは、終りのカットとしては現われることがなく、プログラム全体の ASSUME 文が直後に続くので、ASSERT 文は必要がない。この最初のカットから出る路のための入力カット表明は、プログラムの ASSUME 文を実行することによって得られる。カット実行がプログラムの RETURN 文によって終了するときは、必ずプログラムの本来の PROVE 文が、実行された直後ということになる。そこでこの場合も、ASSERT 文は不要になる。プログラムの PROVE の結果が、カット実行の結果として使える。

以下で述べる正当性証明手法が正しいことは、帰納法の議論を用いることにより、簡単に立証できる。

もし、(最初を除く)各カットに ASSERT 文によって帰納表明をおき、すべてのカットのカット実行がこれらの表明に関して正当であるようにできるならば、プログラムは正当である。もしそのような表明が存在しなければ、プログラムは正当ではない。

カット実行の証明により、カットの入力表明を満たす

1 GCD:

PROCEDURE (M, N);

2 カット;.....ASSUME ( $M > 0$  &  $N > 0$ );

3 DECLARE M, N, A, B INTEGER;

4  $A \leftarrow M$ ;

5  $B \leftarrow N$ ;

6 DO WHILE ( $A \neq B$ );

7 カット;.....ASSERT ( $(A, B) = (M, N) \& A \neq B$ );

8 IF  $A > B$

9 THEN  $A \leftarrow A - B$ ;

10 ELSE  $B \leftarrow B - A$ ;

11 END;

12 PROVE ( $A = (M, N)$ );

13 RETURN (A);

14 END;

図6 帰納表明の入った手続き GCD

ようなプログラム変数の値のいかなる組に対しても(その中には、手続きを実際にこの点まで実行させた結果としての値も、含まれる)、実行は最終的には後続するカットに至り、その後のカットに応じた出力表明が、カット実行の結果得られるプログラム変数の値によって満たされる、ということが立証される。しかし、各カットに対して一つしか表明が置かれていないので、表明はカットに至るすべてのカット実行に対しては出力表明となり、同時にカットから出るすべてのカット実行に対しては入力表明となる。この表明を出力表明として満たす値はすべて、後続の入力表明としてもこれを満たしている。

プログラムの入力表明を満たすような特定のプログラム入力に対して、次のカットに至ったときに計算されている値は、そのカット表明を満たす。さらにその表明は順に次のカットに至ったときに計算されている値がそこでのカット表明を満たすことを、保証する。そして、プログラムの最後の RETURN に至ったときに計算されている値は、プログラムの出力表明を満たす。プログラムのあらゆる入力に対する証明がカットの証明から得られるので、プログラムはすべての入力に対して正しいことになる。

図6のプログラムには、2つのカットがあり、したがって2つのカット実行がある。この2つの実行に対するカット木が、図7と図8に示されている。これらの木の各葉に“検証された”と印字されているので、プログラムは正しい。

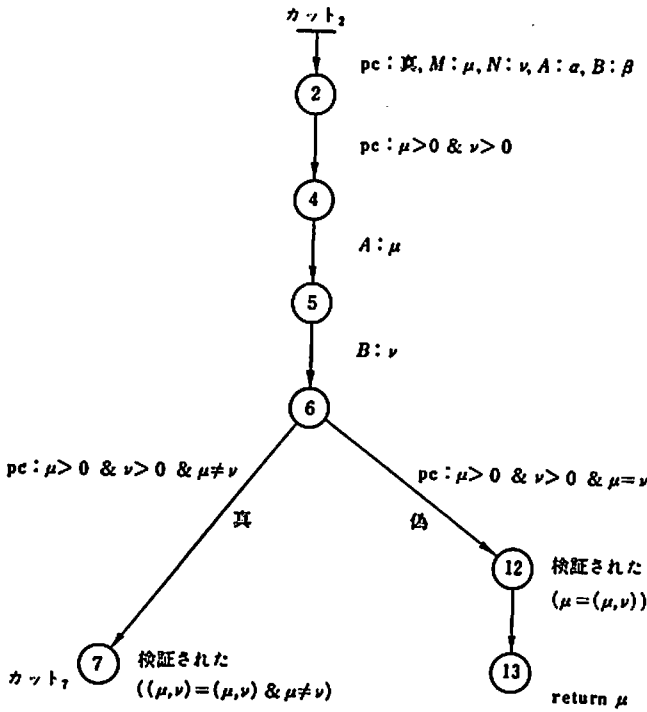


図 7 GCD のカット<sub>2</sub>のカット木

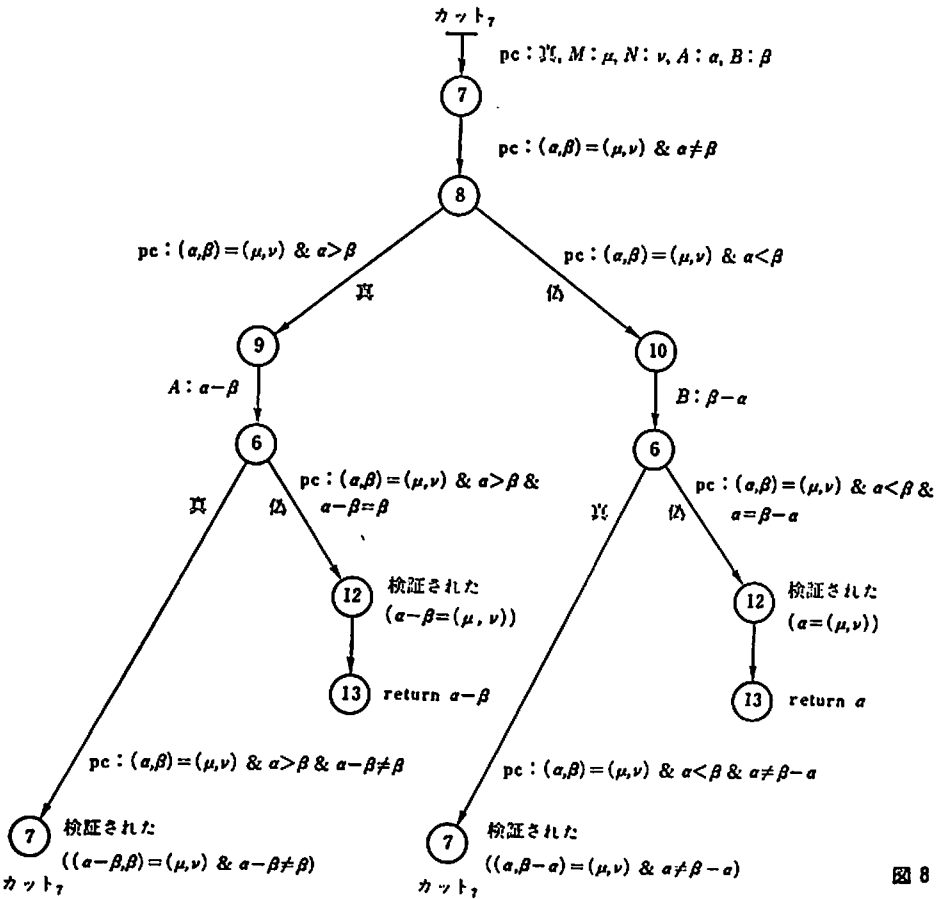


図 8 GCD のカット<sub>7</sub>のカット木

## 5. 手続き

どのような検証技法も、サブルーチンや関数を呼び出すプログラムを効率よく取り扱えなければならない。(ここでサブルーチンと関数を、副手続きと呼ぼう。)

記号実行の考え方は、このような(復帰について準備を整えたうえで)制御の移行、いくつかの変数に対する新たな値の対応づけ、局所手続き変数の生成ないし消滅をとまなう呼出しにも、自然に拡張される。これらの操作はすべて、変数の値が記号式であろうが数であろうが、概念的には同じである。手続き呼出しのあるプログラムの記号実行に対する記号実行木を、正当性の証明をも含め、前に述べたのと同様に考えることができる。

図9に示された、GCD 2 という最大公約数を求める手続きの改訂版を考えてみよう。これは、図1の ABSOLUTE 手続きを呼ぶように変更されている。カット<sub>1</sub> のカット実行は、手続き GCD の図7におけるものと同一である。

カット<sub>1</sub> のカット実行は、図10に示されている。記号実行は、手続き ABSOLUTE に“入りこむ”。ABSOLUTE 内の文の実行から作られた木の節点は、GCD 2 の節点と区別するために三角形で表わされている。

しかしながら、この方法によれば、すべての証明について、副手続きが呼び出されるたびにその性質を証明し直さなければならず、いわば“無から始める”必要がある。

そこで、手続きが正しいことが証明でき、かつその証明を、それを呼び出す手続きの証明において補助定理として使えるような方法がある。次に、そのような方法を述べよう。

手続き(副手続き)が、ある入出力表明の組に関していったん証明されると、手続きの動作につき次のような両立する情報源が存在することになる：1) 実行可能なアルゴリズムとしての手続き自身(手続きのコード本体)；2) 入出力表明に記述されている証明済みの性質。

入出力表明によるプログラムの特徴づけについて、注意すべき2つの重要な点がある。

1) これらは、必ずしもプログラムの動作のすべてではなく、その効果の一部を特徴づけるものである。どんなプログラムでも、結果について何も言明していない真という出力表明に関しては、正当である。出力表明はプログラマによって与えられ、そのプログラマが重要と感じるもののみを含んでいる。

2) 情報は、一般にどう結果を計算するかという記述

1 GCD 2 :

PROCEDURE (M, N) ;

```

2 カット1.....ASSUME (M>0 & N>0) ;
3           DECLARE M, N, A, B, D INTEGER;
4           A←M;
5           B←N;
6           DO WHILE (A≠B) ;
7 カット2.....ASSERT ((A, B)=(M, N)&A≠B);
8           D←ABSOLUTE (A-B) ;
9           IF A>B
10              THEN A←D;
11              ELSE B←D;
12           END ;
13           PROVE (A=(M, N)) ;
14 return   RETURN (A) ;
15           END

```

図9 ABSOLUTE を呼ぶ GCD 2 手続き

ではなく、結果が満足すべき性質を記述するものである。図2のプログラムの出力表明 ((Y=X' | Y=-X') & Y≥0 & X=X') は、そのような Y の値をいかに計算するかを述べていない。入出力表明は、その手続きの結果をいかに計算するかという仕組みを含まず、単にその結果を表わしているだけなので、その手続きを呼び出している手続きの正当性証明には、副手続きのコード本体よりも使いやすい。

副手続きを扱う基本的な手法は、主技法と同じ着想にのっとっている。すなわち、プログラム変数の任意の値を交わすのに、記号を用いるということである。手続きを実行する効果は、呼ぶ側の手続きの変数の値の一部を変更することであり、関数呼出しの場合は、さらに関数値を返すという効果に加わる。手続き呼出しによって値の変更される呼出し側の手続きの各変数に対し、それぞれ1つずつ新たな特定の記号がつけられる。手続きのコード本体を記号実行する代わりに、これらの新しい記号によって、呼出し側の手続きで影響を受ける可能性のある変数の値が、置き換えられる。もし副手続きが正しいと証明されれば、これらの新しい値につき出力表明が成立し、証明に必要なこれらの値の情報が得られる。

この完全なプロセスは、正確には、もとの副手続きから、以下に述べるようなやり方で簡単に得られる“略手続き (abbreviated procedure)”を、普通に記号実行したのものとして、意味づけられる。

1) 手続きの先頭の ASSUME 文を、引数は変えずに PROVE 文に直す。

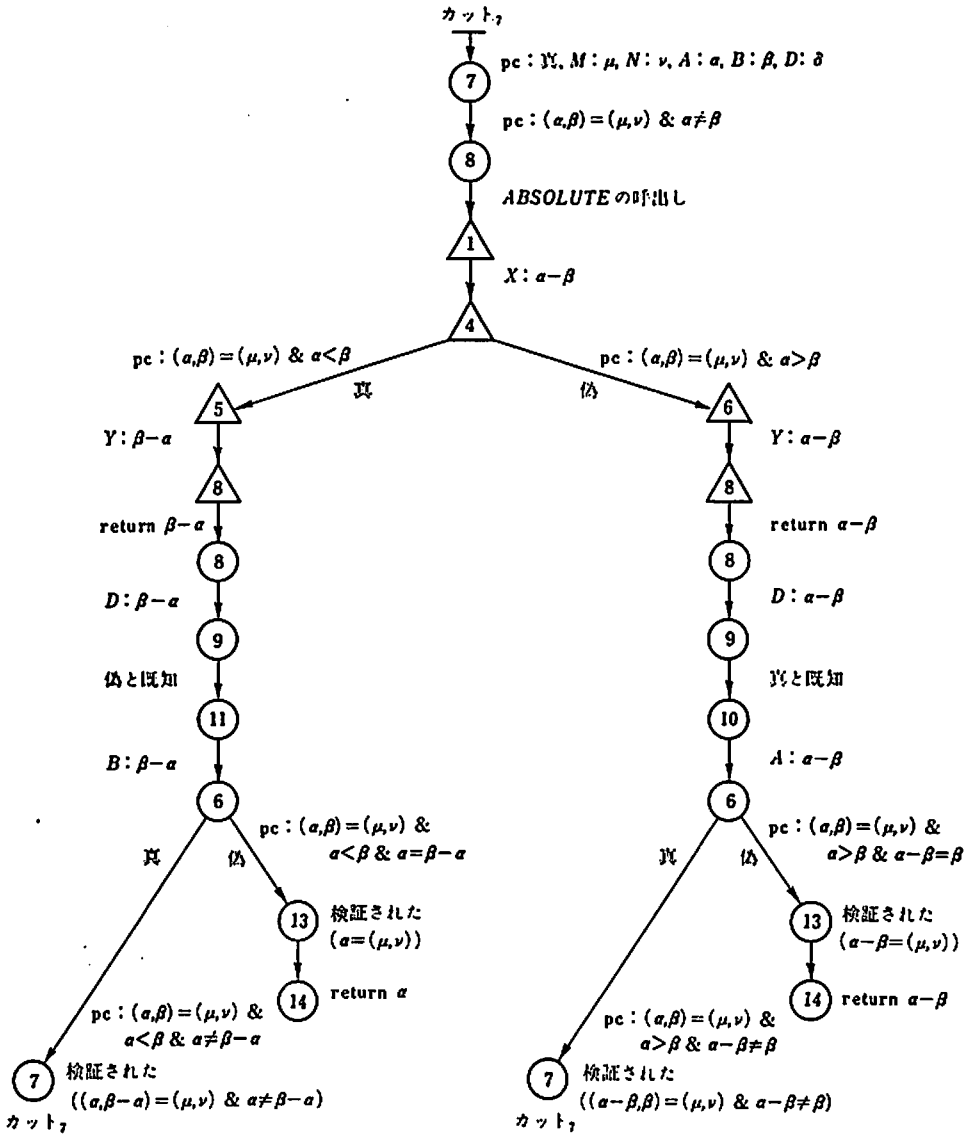


図 10 GCD 2 のカット<sub>7</sub> のカット木

2) 手続きの終りの PROVE 文を、引数は変えずに ASSUME 文に直す。

3) 手続きのコード本体全部を、その手続きで変更を受け得る各変数につき1つずつの、次の形式の代入文の列によって置き換える。

$\nu_i \leftarrow \text{NEWSYMBOL};$

組込み関数 NEWSYMBOL は、それが呼び出されるたびに新しい記号値をその値として返すものとして定義される。

図 2 の手続き ABSOLUTE の略手続きが、図 11 に示されている。

手続き  $Q_1, Q_2, \dots, Q_n$  を (CALL または関数参照によ

```

1 ABSOLUTE:
  PROCEDURE (X);
2   PROVE (真);
3   DECLARE X, Y INTEGER;
4   X ← NEWSYMBOL;
5   Y ← NEWSYMBOL;
6   ASSUME ((Y = X' | Y = -X') & Y ≥ 0 & X = X');
7   RETURN (Y);
8   END
  
```

図 11 略手続き ABSOLUTE

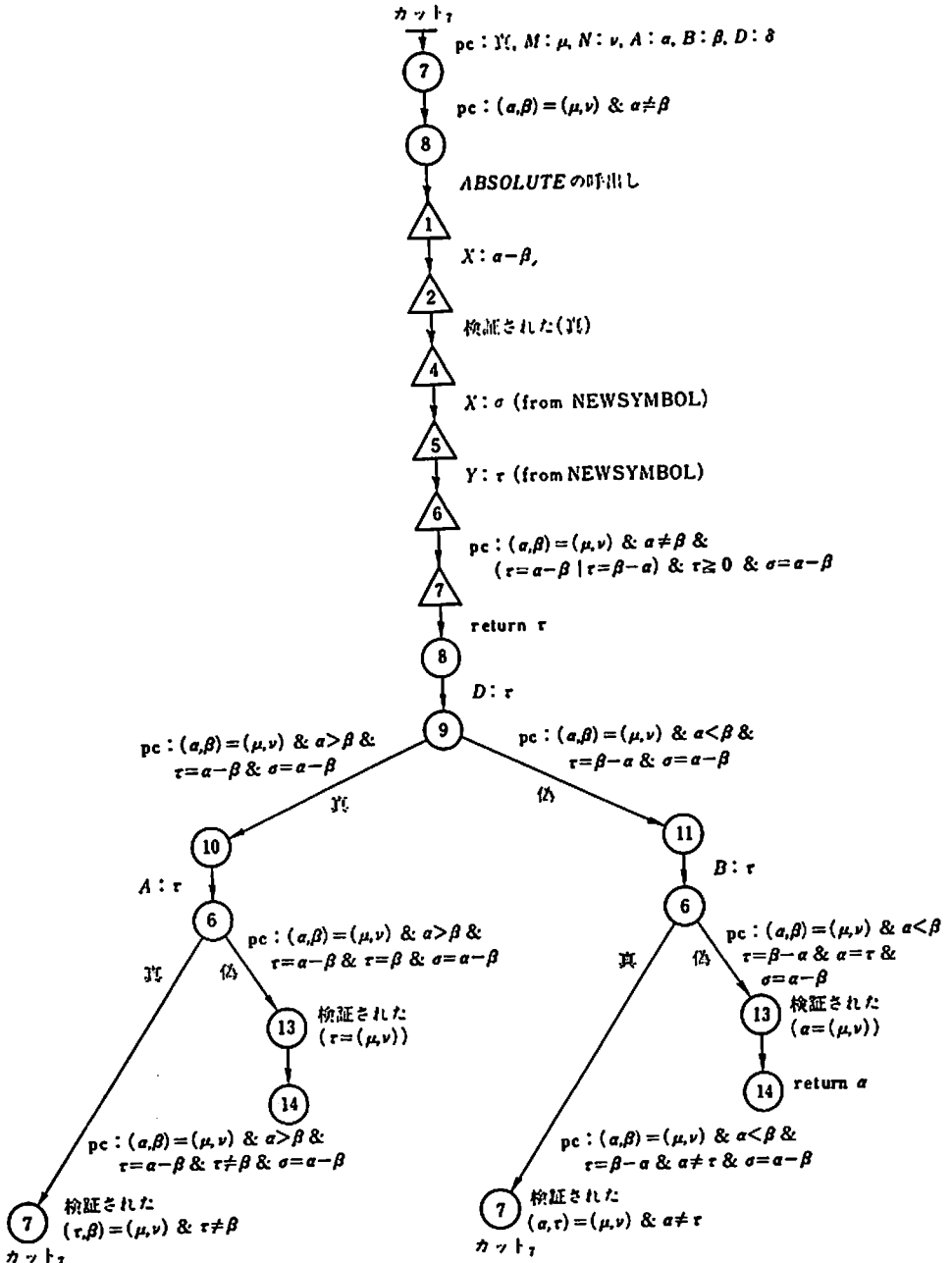


図 12 GCD 2 のカット<sub>1</sub> のカット木 (略手続き ABSOLUTE を使用)

り) 参照するプログラム  $P$  を考えよう。各手続き  $Q_i$  はその ASSUME/PROVE 文に関して正しいことが証明済みだとする。各手続き  $Q_i$  をその略手続きで置き換える。  $P$  には ASSUME/PROVE 文があり、  $P$  の各繰返しは ASSERT 文によってカットされているとしよう。手続き  $Q_i$  の正当性の仮定のもとで、手続き  $P$  がその入出力表明に関し正当であることの証明は、前に 3 節、4 節で述べたのとまったく同様に進む。図 11 の略手続

きを用いた図 9 の GCD 2 のカット<sub>1</sub> のカット実行が、図 12 に示されている。カット<sub>2</sub> のカット実行は、前と変わらず、図 7 にある。これら 2 つのカット実行が、手続き GCD 2 の正当性を証明する。  
 $P$  の路に沿った記号実行のあいだに手続き呼出しの起るたびに、略手続きが呼び出される。  $Q$  がそのような略手続きであるとしよう。  
 $Q$  の証明により、  $Q$  の先頭の入力表明がそれが呼び出

されたときの変数の値によって満たされるなら、出力表明が RETURN 時の値によって満たされることが保証される。したがって、今度はこの手続き呼出しにされる値が、Qのもの ASSUME 文を満たすことを示さなければならない。略手続きをつくる第一ステップ（すなわち ASSUME を PROVE に変える）が、このチェックを首尾よく行なう簡単な鍵を与える。PROVE 文を含めそれに至るまでの略手続きの記号実行が、各ケースでのチェックを果たす。PROVE 文は、そこに書かれた述語が真であるか否かによって、“検証された”あるいは“検証されない”と印字するものと、定義されている。

(PROVE 文で使われる) pc は、記号定数上の条件を記述することにより、準備している（記号的な）計算機の状態の一部と考えられることに、注意すべきである。pc はプログラム変数とは関係がなく、したがって、手続き呼出しによっては影響を受けず、プログラムの実行全体にわたって、広域的な有効範囲をもつ。

もし PROVE 文の実行の結果が“検証されない”となれば、正当性証明は失敗である。これには、いま述べた略手続き内の PROVE 文も含まれる。もしこれが成り立たなければ、手続きの出力表明が成立することが保証されず、この手続きの使用は正しくない。

次に、略手続き Q の記号実行は、Q が接近可能な変数はすべて、新たな特定の記号に設定しなおされる。これらは、手続き Q が実際の実行で計算する新しい値を表わす。Q が関数の場合は、これら新しい記号の一つである RETURN 文の値も、関数値として以下に返される。

略手続きは次に、ASSUME 文（もとの Q の出力条件）を含む。この文の記号実行は、既述の定義どおりに進む。実行はまったく同じ規則に従うのであるが、ASSUME 文と pc の考え方を一般化しておくほうがよいかもしれない。pc の変化は、前に述べたところでは考慮の対象となるケースの微細化（圧縮）であった。この ASSUME 文の実行は、むしろ精緻化をもたらす。略手続きの記号実行は、この ASSUME 文の直前で、この手続きで新しい値を得る可能性のある変数につきすべて新しい記号値を設定している。ASSUME 文の実行は、pc を更新することにより、これらの新しい記号が、副手続きの出力表明を満たすケースに従うという制約をつける。すなわち、この結果これらの記号は、出力表明で特徴づけられる副手続きの変化を表わすことになる。

次に制御は、呼び出したプログラム P に戻る。Q の中でつくられた新しい記号は、Q から返された P の変数の値、および関数手続きの場合は、それから返された値に

なる。P の記号実行は、前と同様に続く。新しい記号の性質についての“知識”が、以降の PROVE 文の実行などで必要になればいつも、pc によって知ることができる。

手続きの出力表明に現われるプライムつき変数は、その手続きへのもとの入力値をその値としてもっているということに、注意されたい。それらは、新しい記号の代入による影響を受けない。一般に、略手続きの記号実行の後では、pc は手続きへの入力（プライムつき変数の記号式の値）と手続きの出力（新たに作りだされた記号）とのあいだを関係づける式を含んでいる。

また、略手続きの記号実行には繰返しもカットもなく、呼び出す手続きのほうでは記号実行の一つの基本的なステップと考えられるということにも注意したい。略手続きの記号実行は、次のような新しい考え方を含んでいる。

- 1) 新しい特定の記号を生成する関数 NEWSYMBOL
- 2) プログラムの正当性は、手続きへの入力を検査するものも含めたすべての PROVE 文の実行が、“検証された”と印字することにより、決定されるという注意。

もちろん、略手続きを実際に作り出す必要はなく、手続き呼出しで生じるのと同じ効果を、単に起こせばよい。

ここでの定義は、副手続きに対する出力表明が、その手続きで変更することが可能で、かつ実際は変更しないようなすべての変数につき、 $X=X'$  という形の文を含まなければならないことを、暗に意味している。もちろん、そのような文は手続き自身の証明において確かめられ、したがって正しいと考えてよい。だから、手続きに関して、出力表明は手続きが何をなすかのみでなく、何をなさざるかについて（たとえば  $X$  を変えない）の文も、含んでいなければならない。もし、プログラミング言語自体が、ある引数につき、それらが“読込み専用 (read-only)”の性質をもつことを規定し保証するならば、文  $X=X'$  はそれらのパラメータに関しては不要である。その場合は、略手続きの定義におけるステップ 3 を、それらの変数が新しい記号で置き直されないように変える必要がある。

読込み専用の引数があれば、次に述べるように、この方法における一つの表記法上のちょっと厄介な問題が解消される。

変更を受けない変数の値は、各手続き呼出しにおいて、名前が変えられる。たとえば、 $X$  が手続きへの引数で、 $\alpha$  という値をもっていたとする。副手続きがそのパ

```

1  EXCHANGE:
   PROCEDURE (X, Y);
2  DECLARE X, Y INTEGER;
3  ASSUME (真);
4  X ← X - Y;
5  Y ← X + Y;
6  X ← Y - X;
7  PROVE ((X=Y') & (Y=X'));
8  RETURN;
9  END;

```

図 13 手続き EXCHANGE

ラメータを  $Y$  と呼んでいて、出力表明に  $Y=Y'$  というものが入っていたとしよう。さらに、略手続き実行中に  $Y$  に対してつくられた新しい記号が、 $\beta$  であるとする。

そうすると、副手続きの実行による、それを呼び出す手続きへの効果は、 $X$  が  $\alpha$  から  $\beta$  へ変化することであるが、 $pc$  は  $\alpha=\beta$  を含むことになる。ここでもし言語の中に、読込み専用であることが既知である変数というものが入り込まれていれば、 $X$  は値  $\alpha$  をずっと保ち続け、 $\beta$  は生成すらされないことになる。

ここで定義され使用されている簡単な言語には、副手続きを含む場合の検証に、ある種の問題をもたらす一つの特徴がある。図 13 に示す、2つの引数の値を、一時的な変数を用いずに、交換する手続きを考えてみよう。

この手続きの、上述した手法による正当性証明は、きわめて単純である。しかしながら、たとえば、

```
CALL EXCHANGE (Z, Z);
```

のように両方の形式パラメータに同じ引数を用いて呼ぶと、その効果は

$$Z \leftarrow Z - Z;$$

$$Z \leftarrow Z + Z;$$

$$Z \leftarrow Z - Z;$$

で、その結果は  $Z$  を 0 とする。(われわれは、PL/I や FORTRAN におけるような、“参照による呼出し (call-by-reference)” を仮定したことを、思い起こされたい。) この場合、出力表明は明らかに満たされない。

そうすると、済んだことになっている証明は、どうなるのか、証明において使われた各カット実行の初めにおいて、各手続き変数は、独立にかつそれらがみな異なる変数だと仮定して、それぞれ単一の記号値に初期化される。この結果、異なる引数による手続き呼出しに対してしか、正当性証明は成り立たないことになる。

この問題には、2つの解法がある。

一つは、同じ引数を、2つ以上のパラメータ (少なく

とも副手続きで読込み専用でないもの) に対応させるような手続き呼出しを、許さないというものである。もう一つは、そのような場合も正しく取り扱えるように、証明方法を拡張することである。正当性証明を行なう際の CALL 文の形を考えれば、この方法は簡単に拡張される。手続き EXCHANGE については、2つの場合がある。すなわち、

1) CALL EXCHANGE (Z, W);

2) CALL EXCHANGE (Z, Z);

この各ケースにつき、記号実行は通常の実行における規則に、忠実に従わなければならない。すなわち、ケース 2 においては、 $X$  と  $Y$  を同じ変数として扱う。その場合証明は、 $Z$  の最終値がゼロとなるので、成功しない。

したがって、この手続きが 2 番目の形で呼ばれるときは、正しいことが証明されないことがわかるので、そのような呼出しは禁止しなければならない。一時的な変数を含むようなもっと典型的な交換のプログラム (たとえば本体が  $T \leftarrow X; X \leftarrow Y; Y \leftarrow T$  であるもの) は、どちらの呼出しの形に対しても、与えられた入出力表明に関し、正しいことが証明できる。

多くのパラメータをもつ手続きでは、2つ以上の引数が一致する組合せの数は、きわめて多くなる。だから、一致のないような証明しやすいケースについて、プログラムを証明し、それからレベルが上のプログラムの証明に必要なような、引数が一致するケースだけを証明するようにすればよいであろう。

## 6. 問題点

プログラムの正当性証明を行なう作業の多くは、長々しく誤りをおかしやすい。

計算機自体にプログラムの証明を行なわせるか、あるいは少なくとも証明を補助させる試みについては、多くの研究がすでになされている [2, 5, 8, 11, 14, 17, 20, 21]。実際、ここで述べたような証明方法は、まず Deutch [5] によってプログラムの自動検証系のために、開発された。

この節では、プログラムの証明を行なううえでの困難な点について簡単なまとめを与える。問題点のいくつかは、その過程を自動化しようとする、とくに重大なものとなる。

正当性証明が楽になるようなプログラミング言語の開発に関し、いくつかの努力がなされている [9, 10]。この目標を追求する際は、その本質的な限界についてわきまえておく必要がある。プログラミング言語は、アルゴ

リズム、それもおそらくある一つの型のアルゴリズムを記述し、またある種のデータを操作する媒介である。この媒介が、アルゴリズムのあいまいな記述を助長して、体裁が悪かったり、必要以上に一般的であったり、きちんと定義されていない言語要素により、形式的 (formal) な解析を困難にしあるいは不可能にすることが、ありうる。しかしながら、アルゴリズムが要求された計算を正しく行なうか否かは、それが記述されている表記法とは独立したことである。

たとえば、単体法が線形制約に従う線形関数を最適化するのに使えるという事実は、きちんとした数学的理論に基づいている。その理論なしには、あるいはその理論を改めて見つけることなしには、完璧なプログラミング言語で書かれたどんなにすっきりした単体法のアルゴリズムも、正しいことは証明され得ない。

記号実行に基づく証明方法は、通常のプログラム実行の拡張と考えられるので、魅力がある。言語の構成要素のある集合に対しては、その通常の実行のもとでの動作を考えることにより、“適切な”証明技法が考え出されることがよくある。記号実行に基づくプログラム検証系 (verifier) の計算機上の作成は、言語の解釈実行系 (interpreter) の作成と、大変似かよっている。

最近の研究の多くは [3, 19, 23], 複雑なデータ構造 (たとえばリスト構造) を取り扱うプログラムを、対象としている。このようなプログラムの難しい点は、データ構造に関して確立された表記法、操作技法や既知の結果といったものが、不足しているためのものである。変数と値についての簡明な数学モデルは、何らかの形で計算機の記憶場所に相当するものが入ってくると、複雑なものとなる。変数は、値をもつ記憶域を参照し、これら (変数、記憶域、値) の関係は、プログラムの実行に応じて変化する。

データ構造の問題と密接な関係にあるのは、汎用的な、融通性のある、読みやすい仕様言語を開発する問題である。プログラムは、その入出力表明に関し正しいことが証明される。この論文での例題は、入出力表明をわかりやすいものをとくに選んだ。

計算機にプログラム検証系を作成するのにとくに重要なことは、必要な数式処理と定理証明の能力を備えることである。プログラムの記号実行には、効率がよく包括的な数式処理システムが必要である。各 PROVE 文の実行は、任意の複雑さをもつ式が真であることを立証しなければならない。プログラムの証明は、効率のよい、対象分野に依存する計算機用定理証明系の研究に、拍車

## 培風館

### ミニコンピュータ システム入門 下

R. H. エックハウス著 / 中西正和訳 A5・240頁 2500円  
基礎からやさしく解き起こし、PDP-11の構成やプログラムを題材にして、コンピュータの原理、オペレーティングシステムやコンパイラ概念を広く解説。ミニコンの使用者やコンピュータシステムを学ぶ人の必読書。

### 情報科学の基礎

上坂吉則・船田哲男・木俣 昇共著  
A5判・240頁・2200円

情報に関する理論の中から、典型的で興味深い通信・信号分析・計算・計画の4テーマを選び、問題の設定の仕方、理論展開の考え方、基本的な結果等、いわば理論の“さわり”を解説し、専門分野へ進むための準備を与える。

### 数学基礎論入門

R. L. グッドステイン著 / 赤 摺也訳  
A5判・208頁・2300円

本書では、過去40年間数理論理学の発展の主役をつとめてきた幾つかのテーマに焦点をあてる。とくに、この間決定的な役割を演じてきたスコーレム、ゲーデル、およびゲンツェンの重要な諸発見について考察する。

### 科学・技術者のための 英文ポリッシュアップ

R. ブランド・新田義孝共著 A5・122頁 1300円  
日本人の間違いやすい点・癖を指摘しながら、英文法・文形の羅列でなく、文章改良のための五つの技術を解説したものである。豊富な例題・演習問題により、自ら文章添削を行ないながら手法をマスターできる。

〒102 東京都千代田区九段南4-3-12  
振替東京4-44725 電話(03)262-5256

をかけている。

最後に論じる大きな問題は、帰納表明をつくることである。

プログラムが与えなければならない入出力表明は、定めるのが難しいことがしばしばある。しかしそれでも、入出力表明が必要なことは、別に人為的には思えない。プログラマは何らかの形で、プログラムが何を意図したものであるかを表わさなければならないからである。しかし、プログラムの繰返しをカットする帰納表明を必ず挿入することは、人為的にみえる。

これらはプログラムの性質を特徴づけるために必要とされるのではなく、プログラムの証明手段に対し帰納を行なう補助として必要とされる。全体のプログラムに対し、次のような大きな定理を定式化することにより、帰納表明の生成を定理証明問題として規定することもできる。すなわち、すべての PROVE 文の結果得られる式が真であるような帰納表明  $P_1, P_2, \dots, P_n$  の存在を証明せよ、という定理である。一般に、これは証明するのは大変難しい定理である。簡単なプログラムに対しては、帰納表明を自動的に生成することもでき、この自動生成ないしは、発見的な方法による、より規模の大きなプログラムへのこの方法の拡張に関する探究が、[12, 22] に報告されている。

## ま と め

この論文は、計算機プログラムが仕様と一致していることを証明するという魅力のある分野への、入門的な紹介を試みたものである。

入出力表明という形式による、プログラムの仕様を記述する一つの流儀を、“正しいプログラム”の定義を可能とするために導入した。それから、プログラムの記号実行について、プログラムのコードと入出力表明が合致していることを立証する一つの方法として、説明を行なった。

すでに提案され、開発されているプログラムの仕様化手法と検証手法の数ある中から、一つをとってそのごく一部を説明したものであることに、注意されたい。しかしこれは、われわれの知る中で最も直観的な方法であり、したがって、将来実り多い発展と利用が最も期待される手法の一つである。

## 謝 辞

ここで述べた考えの多くは、Jerry Archibald, Steve Chase,

Ahmed Chibib, Claus Correll および John Darringer とともに  
行なった研究である EFFIGY システムの成果である。

## 参 考 文 献

- [1] BOYER, R. S.; ELSPAS, B.; AND LEVITT, K. N. "SELECT — A formal system for testing and debugging programs by symbolic execution," *Internat. Conf. on Reliable Software*, 1975, ACM, New York, 1975, pp. 234-245.
- [2] BOYER, R. S.; AND MOORE, J. S. "Proving theorems about Lisp functions," *J. ACM* 22, 1, (Jan. 1975), 48-59.
- [3] BURSTALL, R. M. "Some techniques for proving correctness of programs which alter data structures," *Machine intelligence 7*, D. Michie (Ed.), American Elsevier, New York, 1972.
- [4] CLARKE, LORI, *A system to generate test data and symbolically execute programs*, Report #CU-CS-060-75, Univ. of Colorado, 1975.
- [5] DEUTSCH, L. P. "An interactive program verifier," PhD Dissertation, Dept. Computer Science, Univ. of Calif. Berkeley, 1973, Xerox PARC Report CSL-73-1, Palo Alto, Calif.
- [6] ELSPAS, B. et al., "An assessment of techniques for proving program correctness," *Computing Surveys* 4, 2 (June 1972), 97-147.
- [7] FLOYD, R. W. "Assigning meanings to programs," in *Proc. Symposium Applied Math.*, Vol. 19, American Mathematical Society, Providence, R. I., 1967, pp. 19-32.
- [8] GOOD, D. I.; LONDON, R. L.; AND BLEDSOE, W. W. "An interactive program verification system," *IEEE Trans. on Software Engineering* 1, 1, (April 1975), 59-67.
- [9] GOOD, D. I.; AND RAGLAND, L. C. "Nucleus—a language of provable programs," in *Program test methods*, W. Hetzel (Ed.), Prentice-Hall Inc., Englewood Cliffs, N. J., 1973, pp. 93-117.
- [10] HOARE, C. A. R.; AND WIRTH, N. "An axiomatic definition of the programming language PASCAL," *Acta Informatica* 2, (1973), 335-355.
- [11] IGARASHI, S.; LONDON, R. L.; AND LUCKHAM, D. C. "Automatic program verification I: a logical basis and its implementation," *Acta Informatica* 4, (1975), 145-182. Also in USC Information Sciences Institute Report ISI/RR-73-11, May 1973.
- [12] KATZ, S. M.; AND MANNA, Z. "A heuristic approach to program verification," in *Proc. Third Internat. Joint Conf. on Artificial Intelligence*, SRI Publications Dept. Stanford Calif., 1973, pp. 500-512.
- [13] KING, J. C. "Proving programs to be correct," *IEEE Trans. on Computers* C-20, 11, (Nov. 1971), 1331-1336.
- [14] KING, J. C. "A program verifier," PhD Dissertation. Carnegie-Mellon Univ., Pittsburgh, Pa., 1969.

- [15] KING, J. C. "A new approach to program testing," in *Internat. Conf. on Reliable Software*, 1975, ACM, New York, 1975, pp. 228-233. Also appears in *Programming methodology, lecture notes in computer science*, 23, Springer-Verlag Inc., New York, 1974, pp. 278-290.
- [16] KING, J. C. "Symbolic execution and program testing," *Comm. ACM* 19, 7, (July 1976), 385-394.
- [17] LONDON, R. L. "The current state of proving programs correct," in *Proc. of ACM Annual Conf.*, 1972, ACM, New York, 1972, pp. 39-46.
- [18] MANNA, Z. *Mathematical theory of computation*, McGraw-Hill Book Co., New York, 1974.
- [19] OPPEN, D. C., AND COOK, S. A. "Proving assertions about programs that manipulate data structures," in *Seventh Annual ACM Symposium on Theory of Computation*, 1975, ACM, New York, 1975, pp. 107-116.
- [20] SUZUKI, N. "Automatic program verification II: verifying programs by algebraic and logical reduction," in *Internat. Conf. on Reliable Software*, 1975, ACM, New York, 1975, pp. 473-481.
- [21] TOPOR, R. W. "Interactive program verification using virtual programs," PhD Dissertation, Univ. of Edinburgh, Edinburgh, Scotland, 1975.
- [22] WEGBREIT, B. "The synthesis of loop predicates," *Comm. ACM* 17, 2, (Feb. 1974), 102-112.
- [23] WEGBREIT, B.; AND SPITZEN, J. M. "Proving properties of complex data structures," *J. ACM* 23, 2 (April 1976), 389-396.

[完]

(訳 たいてつお 三菱総合研究所)

## お詫びと訂正

6月号に掲載いたしました「プログラムの正当性証明入門1」に下記の誤りがありましたので、お詫びして訂正いたします。

誤 正

p.24 関注 CACM → acm computing surveys

## I/O 別冊『徹底研究シリーズ』

B5判 定価1,900円(〒200)

I/O別冊⑦=6月上旬発売!

## マイコンゲーム徹底研究

- BASICからマシン語まで、マイコンを使いこなしたいあなたのためのゲーム集
- マシンは…APPLE, PET, TK-80BS, ペーシックマスター-LKIT-16, M110, SDK-85……etc.

## ■好評既刊

## I/O別冊①マイコン徹底研究

☆M6800を中心にマイコンのつくり方からTVゲームの作り方まで、ていねいに解説

☆キャラクタ・ディスプレイ、キーボード、放電プリンタA/D, D/A, フロッピーなど周辺についても充実!

## I/O別冊②TVゲーム徹底研究

☆LSIゲーム、TTLゲームからマイコンゲームまで徹底的に解説

## I/O別冊③BASICゲーム徹底研究

☆BASICの入門から応用までわかりやすく解説

☆ゲーム=はさみ将棋/Cat & Mouse/Submarine/π e /バイオリズム…

## I/O別冊④マシン語徹底研究

☆マシン語をまったく知らない人が自分でプログラムできるようにする入門書

☆CPU=8080, 6800, 6502, Z80

## I/O別冊⑤RANDOM BOX(ランダム・ボックス)

☆全国マイコン・ファンアイデア集

☆マイコンのハードからソフトまで114編を収録!

## I/O別冊⑥BASICゲーム徹底研究②

☆レベル2BASICを使いこなしたいあなたのためのプログラム集!

Computer fan NO.2  
コンピュータファン

6月中旬発売!! B5判 定価650円(〒160)

☆驚異のマイコン・ソフト開発ツールとマイコン・パズル

☆主要プログラム収録レコード付!!

## ■好評既刊

コンピュータファンNO.1 420円(〒160)

☆高速BASICコンパイラ、アセンブラ

●マイコンコンピュータの専門店 I/O (アイ・オー)

●B5判 平均170頁 毎月25日発売 ¥380

●定期購読

●半年 ¥2,300 ●1年 ¥4,300

●送料別添付、現金送付、定額小為切のいずれでも可

東京・代々木

工学社

☎(03)375-5784代

振替口座 東京5-22510

〒151 東京都渋谷区代々木1-37-1 ぜんらくビル