ソフトウェア工学　80－20
（１９９１．７．１９）

プログラム論理図の形式的表現

大場　克彦　　金戸　孝夫

島津製作所　東京研究所

難しい理論に精通していなくても、一定の手順に従って設計を進めると、プログラムの形式的仕様が得られる手法として、HOS（Higher Order Software）がある。筆者らはHOSの手法について評価を行ったところ、この手法で作成される設計書が冗長になるという結果を得た。

筆者らはHOSの方法論を参考にしながら、簡潔な記述が出来るように記述法の改善を行い、プログラム論理図の形式表現を試みた。このプログラム論理図は、木構造で表現されている。

Formal Expression of Program Logic Diagrams

Katsuhiko Ohba　Takao Kaneto

SHIMADZU CORPORATION

The methodology of HOS(Higher Order Software) make possible to get us the formal specifications without to know the theory that HOS is based on. We estimated HOS, and get the conclusion that the specifications described by HOS mesthdology is too lengthy.

We attempted to improve the expression of HOS to make possible to describe the specifications briefly. Based on the result of this attemption, we developped the formalized expression of program logic diagrams. These program logic diagrams are presented by the tree structured diagrams.

1　はじめに

ソフトウエア開発において、要求仕様のあいまいさが手戻り作業を生じ、工程遅れや開発コストを高くする原因になっていることが指摘されている[1]。しかし、ソフトウエア開発の現状をみると、要求分析段階だけでなく設計段階において作成される資料にもあいまいさが含まれている。例えば、設計段階で作成されるフローチャートや木構造図のようなプログラム論理図について考えてみても、１つのプログラム論理図から１つのソースプログラムが一意に定まるというわけではない。信頼性の高いプログラムを開発するためには、設計書の形式性を高めることが重要であることが指摘されている[2]。プログラムの設計資料の形式性を高めるために、形式仕様記述法の研究成果を利用することが有効であると思われる。形式的仕様記述法の中では、等式論理に基づく代数的仕様記述法が、比較的分かりやすく思われた。しかし、実際には、逐次処理型の実用レベルのプログラムの仕様を、平均的なプログラマが代数的仕様記述法を用いて書くのは困難である。その原因は

①基礎となる数学理論を理解しないと仕様が書けないが、平均的なプログラマがこれを理解するのは困難である。

②理論にしたがって、厳密に仕様を記述すること自体が難解な作業である。例えば、公理を表す等式集合をどの様な基準に従って導きだしたらよいのか分からない。

などである。これは、筆者らが代数的仕様に接したときの感想であるが[3]、代数的仕様に限らず、形式的仕様記述法全般についても似たようなことがらが指摘されている[4][5]。

形式的仕様記述法が、ソフトウエア開発の現場で利用できるようになるためには、難解な基礎理論を意識しなくても、定められた手順に従って記述すれば、厳密であいまいさのない形式的な仕様が得られることが必要である。さらに、記述が簡潔であることが必要である。簡潔な記述は、記述を容易にし、迅速に仕様を作成することを可能にすると共に、記述された仕様を理解しやすくする上で必要である。

このような観点から形式的仕様記述法について検討し、プログラム論理図の形式的表現を可能にしたので報告する。

2　設計思想

基礎となる数学理論を知らなくても、ある一定の規則にしたがって仕様を記述すれば、あいまいさのない仕様が記述できることが、まず必要である。このような観点から調査を進め、ＨＯＳ（Ｈｉｇｈｅｒ　Ｏｒｄｅｒ　Ｓｏｆｔｗａｒｅ）[6][7][8]を発見した。これを試用した結果、仕様の簡潔性という点で問題があり、これを改善する方向で検討を行った。まず、筆者らの手法の出発点となったＨＯＳの手法について紹介し、次にその長所と問題点について述べ、最後に問題点の改善の方向について述べる。

2．1　ＨＯＳ

ＨＯＳの特徴は、大きく分けて２つある。１つは、基礎となる理論に関する知識がなくても完全かつ無矛盾な公理（注１）を、一定の手順に従えば導き出すことが出来る点である。第２は、コンピュータ支援系を持っていることである。コンピュータ支援系により、公理の完全性、無矛盾性の検証とソースプログラムの自動生成を行っている。

注１：ＨＯＳでは仕様を公理と呼んでいる。

ＨＯＳの公理を導く手順について述べる。ＨＯＳでは、プログラム及びプログラムを構成する操作を、多入力多出力の関数として表現する。次に、プログラムを表す関数を２つの操作に分割する。分割した操作がこれ以上分割できない操作であればそこで分割をやめる。そうでなければ、さらにその操作を２つの操作に分割する。末端の操作が、それ以上分割できない操作で表現されるまで、上に述べた手順を繰り返す。このようにして得られる公理は、２進木として表現される。

これ以上分割できない関数として、基本関数、再帰関数、外部関数、定義済み関数の４種類がある。基本関数は、基本データ型に対する操作である。再帰関数は、反復処理を可能にするための関数である。基本関数および再帰関数は、ＨＯＳであらかじめ用意してある関数である。外部関数は、ＨＯＳの方法によらずに開発したプログラム（関数）で再利用可能なものである。定義済み関数はＨＯＳの方法により開発したプログラム（関数）である。

階乗の計算をするプログラムについて、公理（図１［ａ］）を導く手順を示す。階乗の計算をするプログラムをＦＡＣＴＯＲＩＡＬとし、入力が変数ＩＮに入れて渡され、計算結果が変数Ｆに入れて出力されるものとする。まずＦＡＣＴＯＲＩＡＬをＩＮＩＴとＦＡＣＴという２つの操作に分割する。ＩＮＩＴは初期化を行う部分で、ＦＡＣＴは階乗の計算を実際に行う部分である。

図１において、幹に当たる関数を親関数、枝に当たる関数を子関数と呼ぶ。ＦＡＣＴＯＲＩＡＬ、ＩＮＩＴ、ＦＡＣＴの関係において、ＦＡＣＴＯＲＩＡＬが親関数で、ＩＮＩＴとＦＡＣＴが子関数である。親関数と子関数の関係を規定する７つの規則が定められている。親関数を２つの子関数に分割したとき、どの規則を適用するかを指定する。この７つの規則は、制御とデータの受渡しの規則を定めたものである。ＦＡＣＴＯＲＩＡＬ、ＩＮＩＴ、ＦＡＣＴの関係では、ＣＪという規則を適用することを指定している。

このとき親関数の動作は、右側の子関数を実行した後、左側の子関数を実行した時の動作と等価であることを意味している。また、データの受渡しでは、

①右側の子関数（右子関数と以後呼ぶ）は少なくとも１つの入力を親関数から受け取らなくてはならない。

②左側の子関数（左子関数と以後呼ぶ）は親から入力を受け取ってもよいし、受け取らなくてもよい。

③左子関数は、右子関数の出力から少なくとも１つ、入

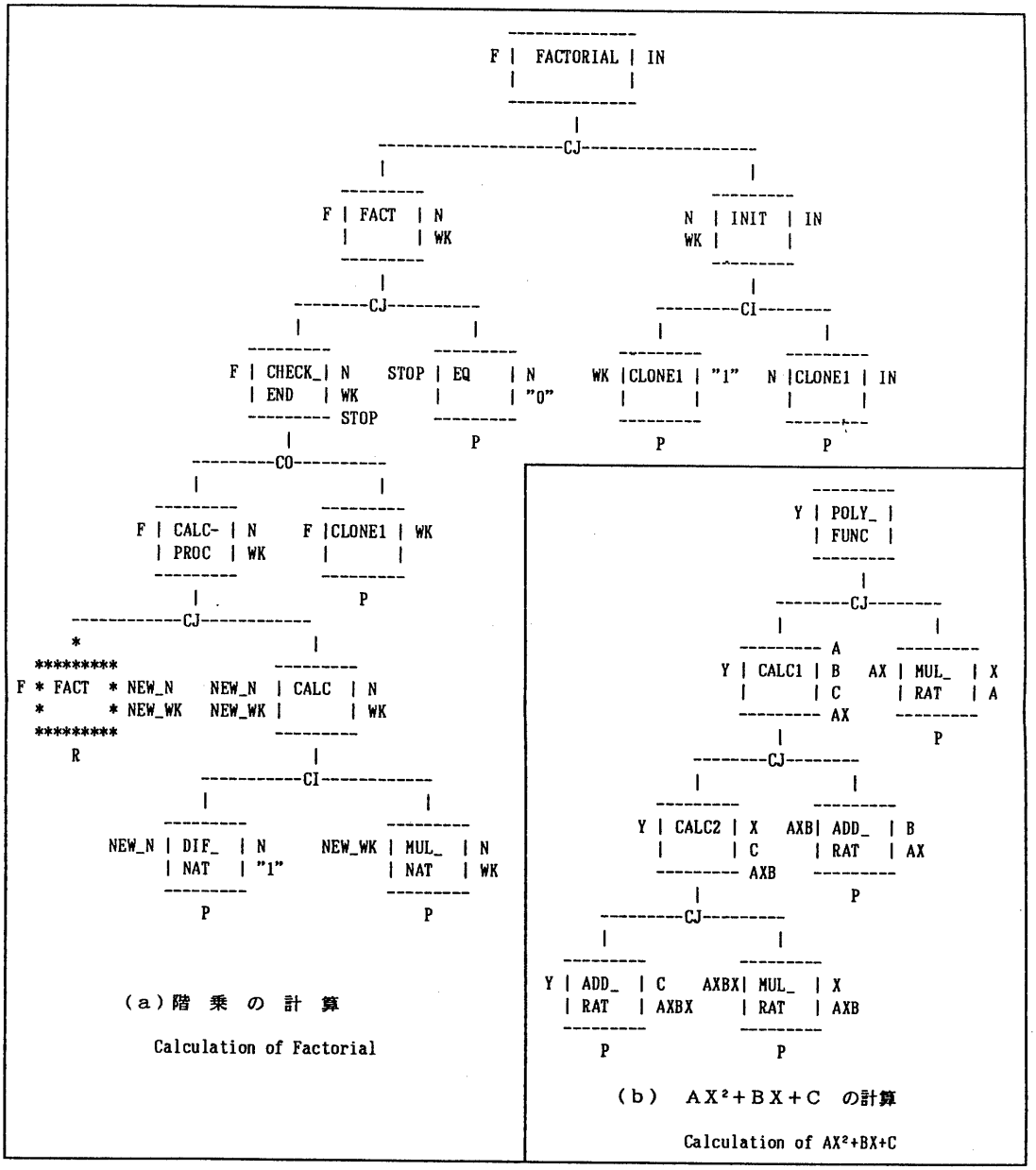

図1　HOSの手法による仕様の記述例
Exsample of Specifications by HOS

力を受け取らなくてはならない。
④親関数の出力は、左子関数の出力の中になければならない。

FACTORIAL、INIT、FACTの関係では、INITがFACTORIALからINを受け取り、NとWKを出力する。FACTはNとWKを入力として受け取りFを出力として、FACTORIALに渡す。

次にINITを2つの操作に分ける。この操作は、どちらもCLONE1という基本操作である。図1において、CLONE1の下にあるPが基本関数を表す記号である。INITの右子関数のCLONE1は、その親関数からINを受け取って入力とし、Nを出力している。このNが親関数に返される。左子関数のCLONE1は、定数1を入力とし、WKを出力としている。このWKも

親関数の出力となっている。ＣＬＯＮＥ１は入力の値をそのまま出力する関数である。この時点でＮにはＩＮの値が、ＷＫには１が設定されている。ＩＮＩＴとその２つの子関数の間にはＣＩという関係を適用することを指定している。ＣＩという関係では、右子関数、左子関数の順に実行されている。また、データの受渡し規則は次のようになる。

①子関数は、親関数の任意の入力を自分の入力とすることが出来る。

②親関数の出力は、どちらかの子関数の出力の中になければならない。しかし、２つの子関数の出力の中に同じ出力があってはならない。

ＣＬＯＮＥ１は基本関数なのでこれ以上分割は行わない。

ＦＡＣＴＯＲＩＡＬの右子関数ＩＮＩＴの分割が終了したので、左子関数ＦＡＣＴを２つの子関数で表す。ＦＡＣＴの右子関数ＥＱは、２つの入力値が等しければ真を出力し、等しくなければ偽を出力する基本関数である。この場合にはＮの値が零ならＳＴＯＰの値が真、零でなければ偽になる。左子関数ＣＨＥＣＫ＿ＥＮＤは、ＦＡＣＴの処理を終了させるかどうかを判定する関数である。この関数は入力として親関数からＮとＷＫ、右子関数ＥＱからＳＴＯＰを受け取り、Ｆを出力として親関数ＦＡＣＴに返す。

ＣＨＥＣＫ＿ＥＮＤはさらに２つの子関数に分割されている。ここではＣＯという関係が指定されている。ＣＯは、選択という制御構造とそのときのデータの受渡しを規定する関係である。ＣＯという関係の時は、親関数の最後の入力データの値が真なら右子関数が実行され、偽なら左子関数が実行される。ＣＨＥＣＫ＿ＥＮＤの場合には、ＳＴＯＰが真ならＣＬＯＮＥ１が、偽ならＣＡＬＣ＿ＰＲＯＣが実行される。ＣＬＯＮＥ１は、親関数からＷＫを入力として受け取り、それをそのままＦにコピーしてＣＨＥＣＫ＿ＥＮＤに返す。ＣＡＬＣ＿ＰＲＯＣはＣＨＥＣＫ＿ＥＮＤからＮとＷＫを入力として受け取り、Ｆを出力として返す。ＦＡＣＴはＣＨＥＣＫ＿ＥＮＤの右子関数が実行されたとき終わりになる。ＣＯという関係のデータの受渡し規則は次のようになる。

①各子関数の入力は、親関数の入力の中に含まれている。

②各々の子関数の出力は、親関数の出力を単独で全て含んでいる。

ＣＡＬＣ＿ＰＲＯＣを、ＣＡＬＣとＦＡＣＴという２つの関数に分割する。右子関数ＣＡＬＣは、ＣＡＬＣ＿ＰＲＯＣからＮとＷＫを入力として受け取り、ＮＥＷ＿ＮとＮＥＷ＿ＷＫを出力する。この２つの出力が、そのまま左子関数ＦＡＣＴの入力になる。このＦＡＣＴは再帰関数である。ＦＡＣＴの下にあるＲが再帰関数を表す記号である。再帰関数ＦＡＣＴは、ＮをＮＥＷ＿Ｎ、ＷＫをＮＥＷ＿ＷＫと置き換えて、前に出てきた同じ名前の関数ＦＡＣＴを再び実行することを意味する。

ＣＡＬＣはさらにＭＵＬ＿ＮＡＴとＤＩＦ＿ＮＡＴという基本関数に分割される。ＭＵＬ＿ＮＡＴは、２つの入力の積を出力する基本関数である。ここでは、ＮとＷＫの積をＮＥＷ＿ＷＫに出力している。ＤＩＦ＿ＮＡＴは、第一の入力の値から第二の入力の値を引いた結果を出力する基本関数である。ここではＮの値から１を引いた値をＮＥＷ＿Ｎとして出力している。さて、これで末端の操作がすべて基本関数と再帰関数で表現されたので、公理は完成したことになる。このようにして得られた公理は、一種のプログラム論理図になっている。

基本関数は、すべてライブラリにその名前と入出力データの型が登録されている。公理の中に現れるデータは、最終的に基本関数に渡されるので、そこでデータの型が決定される。一般に同じデータがいくつかの基本関数に受け渡されていくので、その間にデータの型の不一致が生じると、公理に論理的な矛盾が生じたことになる。このような矛盾がないかを検証している。

筆者らはＨＯＳを試用した結果

（１）基礎理論を知らなくても形式的仕様が記述できる。

（２）仕様を階層的な図式で表現している。

（３）コンピュータ支援系を持っており、自動的に仕様の完全性、無矛盾性の検証を行った後、ＦＯＲＴＡＲＡＮやＣのソースコードが生成される。

などの長所を有している。しかし、プログラムの経験者にとっては、次のような問題点があった。

（１）仕様記述に時間がかかる。実際に、ＦＯＲＴＲＡＮやＣでプログラミングするのに比べ３倍以上のの時間を要した。

（２）記述された仕様が理解しにくい。ＦＯＲＴＲＡＮやＣで書かれたプログラムの方が理解しやすい。

ＨＯＳの最大の問題点は、仕様が冗長になることである。これが仕様の記述に時間を要し、その内容を分かり難くしている最大の原因である。図１（ａ）に階乗、図１（ｂ）に２次式の仕様を示す。この例にみられるように、仕様を冗長にしているのは、

①式は検証の対象にならないので式が直接記述できない。

②プログラム言語で簡単に表現できる制御構造を表現するのに、複雑な表現をしなければならない。

③１つの操作を、２つの操作で表現することを基本原理としているため、本来は必要でない操作が含まれる。

などが原因である。

第２の問題点は、関数間の関係を指定しなければならないことである。この関係を指定するのに時間がかかると共に、この指定ミスによる誤りがかなり発生した。これが仕様記述に時間を要する２番目の原因になっていた。

２．２　設計の基本方針

以上の調査結果を基に、プログラム論理図の記述法の設計に当たって

（１）理論を意識しなくても、厳密な形式性を有するプログラム論理図が作成できる。

（２）コンピュータを利用して、プログラム論理図の矛盾の検出ができる。

（３）プログラム論理図からソースプログラムを自動的

に生成できる。
（４）ＨＯＳのプログラム論理図より、記述し易く理解し易い。
の４点を、基本的な目標とした。
（１）については、まずプログラムを文を用いて記述するものとし、文の間の等価関係の定義の集合としてプログラムを表すと共に、プログラムを文の等価関係の集合で表すための手順を定め、この手順に従って等価関係を導けば、曖昧さのない仕様が得られる様にした[9]。次に、この等価関係の集合を木構造を有するプログラム論理図としてで表現することを試みた[9][10]。これは、ＨＯＳの考え方と同じである。
（２）と（３）を達成するために、プログラム論理図に厳密な形式性を与えることを試みた。
（４）については、ＨＯＳの記述の冗長さを削り、記述がしやすく、かつ分かりやすくするため、次に述べるような機能の追加や記述方法の改善を行った。
①数式の記述を許す。
②１つの文を２つ以上のの文で表現することを許す。
③制御構造に関する情報とデータの受渡し規則を分離し、
・制御構造については、選択や反復を記述する文を導入すると共に、この文の中で選択や反復の条件を論理式として記述することを許す。
・データの受渡し規則については、規則をもっと単純化しプログラム論理図の中でいちいち指定しなくてもよいようにする。
④日本語が使える。

３　文

プログラム論理の記述に厳密な形式性を保証するためには、そこで使われる用語の構文と意味を定義しておく必要がある。ここでは、文を使ってプログラム論理を記述する。ここで、文は入力に対して操作を加えると出力を出すものとして定義する。このように定義すると、文は多変数入力、多変数出力の関数と見なすことができる。ただし、副作用を許すという点で純粋の関数とは異なる。プログラムも１つの文である。
ここでは、文を形式的に表現するため、

操作（入力１、・・、入力ｋ）
→（出力１、・・、出力ｍ）　　［１］

と表す。ここで、操作、入力変数、出力変数には日本語表現が許される。
次に文を、未定義文と既定義文に分類する。未定義文は、実現方式が定義されていない文である。既定義文は、実現方式が定義されている文である。さらに、既定義文を基本文と定義文に分ける。基本文は、高級言語で用意されている基本データ型に対する文であり、高級言語の文がこれに対応する。定義文は、基本データ型や既に定義されているデータ型を組み合わせて新しく作ったデータ型に対する文で、基本文および自分より前に定義されている定義文を用いて新たに定義した文である。定義文は、１つのプログラムである。

３　プログラム論理の設計手順

上に述べた文を使って、次の手順でプログラム論理の設計を行う。
（１）プログラムの名前と入力データ、出力データを、［１］の書式にしたがって書き表す。プログラムの名前は、その機能が分かるようにつける。
（２）プログラムの機能を実現するために、どの様な処理を施せばよいかを手順として表す。ここでは、文の操作だけを順番に書き、その実現手段については考えない。すなわち、未定義文として処理を記述する。
（３）各未定義文への入出力を決め、［１］の書式で表す。
（４）各未定義文が実現されれば、プログラムの機能が実現されることを確かめる。
（５）未定義文のうち、既定義文で置き換え可能なものは置き換える。
（６）データの受渡し規則を満足していることを確かめる。
（７）全ての未定義文が、既定義文で表現可能になるまで以下の手順を繰り返す。
①未定義文を、新しい未定義文を使って記述する。
②新しい未定義文が実現されれば、元の未定義文が実現されることを確かめる。
③新しい未定義文のうち、既定義文で置き換え可能なものは置き換える。
④データの受渡し規則を満足していることを確かめる。
以上の手順で得られたものをプログラム論理仕様と呼ぶ。

４　プログラム論理仕様の等価表現

４．１　順次構造の等価表現

いまプログラムをＰ、未定義文をＫｉ、既定義文をＳｉで表す。３の手順で得られるプログラム論理仕様は、次のような等価関係で表される。
まず、（１）ー（４）の手順により、プログラムと未定義文の等価関係が得られる。これを［２］式で表す。

Ｐ≡Ｋ１；Ｋ２；Ｋ３；・・；Ｋｎ　　［２］

ここで、Ｋ１；Ｋ２；Ｋ３；・・；Ｋｎは、未定義文Ｋ１、Ｋ２、Ｋ３、・・、Ｋｎを、この順に施すことを意味する。［２］式は、プログラムの動作が文Ｋ１、Ｋ２、Ｋ３、・・、Ｋｎをこの順番に施したときの動作と同じであることを表す。ここで、同じ動作をする文あるいは文の列は等価であると定義し、記号”≡　”で表す。
（５）の操作により、［２］式の中の未定義文と既定義文の等価関係を表す式の集合が得られる。例えば、Ｋ１がＳ２、Ｋ３がＳ５と等価であるとき、次の等価関係式が得られる。

Ｋ１≡Ｓ１　　［３］
Ｋ３≡Ｓ５　　［４］

（２）式の中の既定義文に置き換えられない未定義文のそれぞれについて、新しい未定義文を用いて等価関係を定義する。例えば、Ｋ２を次のように定義する。

Ｋ２≡Ｋ２．１；Ｋ２．２　　［５］

ここで、Ｋ２．１、Ｋ２．２は、新しい未定義文である。いま、Ｋ２．１がＳ２、Ｋ２．２がＳ３と等価関係にあれば、

Ｋ２．１≡Ｓ２　　［５］

Ｋ２．２≡Ｓ３　　［６］

と表される。以下同様の手順で、未定義文を新しい未定義文を使って定義して行くと、プログラム論理仕様を表す等価関係式の集合が得られる。この等式関係式は、左辺の項を右辺の項に書き換える項書換え規則と見なすことが出来る。プログラム論理仕様が意味を持つためには、この項書換え規則を適用したとき、プログラム論理仕様の中のすべての未定義文が既定義文の列に置き換えられることが必要である。

４．２　選択構造

いま、未定義文Ｋ０［Ｃ］を考える。Ｋ０［Ｃ］は条件Ｃが満足されたとき未定義文Ｔを、満足されないとき未定義文Ｆを実行するものとする。ＩＦ［Ｃ］という基本文を導入し、Ｋ０［Ｃ］を［７］式の等価関係で表す。

Ｋ０［Ｃ］≡ＩＦ［Ｃ］；Ｔ；Ｆ　　［７］

Ｃは真か、偽のいづれかの値をとる。ＩＦ［Ｃ］は、Ｃが真ならＴを、偽ならＦを実行する基本文である。これを表す等価関係が［８］［９］である。

ＩＦ［真］；Ｔ；Ｆ≡ＩＦ［真］；Ｔ　　［８］

ＩＦ［偽］；Ｔ；Ｆ≡ＩＦ［偽］；Ｆ　　［９］

［７］［８］［９］の等価集合から、［１０］［１１］の等価関係が得られ、これがＫ０［Ｃ］の意味を表している。

Ｋ０［真］≡ＩＦ［真］；Ｔ　　［１０］

Ｋ０［偽］≡ＩＦ［偽］；Ｆ　　［１１］

４．３　反復構造

条件Ｃが満足されている間、文Ｋを反復実行する未定義文をＬ［Ｃ］とする。Ｌ［Ｃ］を［１２］の等価関係で定義する。

Ｌ［Ｃ］≡Ｗ［Ｃ］；Ｋ；Ｒ　　［１２］

ここでＣは、真か偽のいずれかの値をとる。Ｗ［Ｃ］は、Ｃが真のときＫとＲを実行し、偽のときＫとＲをスキップする基本文である。Ｒは、反復構造の終端を表す記号に対応し再帰操作を表す。

この意味を［１３］［１４］の等価関係で定義する。

Ｌ［真］≡Ｗ［真］；Ｋ；Ｒ　　［１３］

Ｌ［偽］≡Ｗ［偽］　　［１４］

再帰操作Ｒは、［１５］等価関係で定義する。すなわち、再帰操作Ｒの実行は、Ｌ［Ｃ］の実行と等価であると定義する。

Ｒ≡Ｌ［Ｃ］　　［１５］

４．４　データの受渡し規則

未定義文の意味は、１つ以上の文を用いて記述される。ある１つの未定義文の意味を定義した文の集まりを複文と呼ぶ。複文の中の個々の文を記述文と呼ぶ。

未定義文と記述文の間に、次のデータの受渡し規則を設ける。

規則１

①未定義文の入力データは、いずれかの記述文の入力データとして使われていなければならない。

②未定義文の出力データは、いずれかの記述文の出力データの中になければならない。ただし、記述文の出力データがすべて、未定義文の出力データになる必要はない。

③異なる記述文が同じデータを出力している場合には、後から実行される記述文の出力データが、未定義文の出力データとなる。

④記述文の入力データは、未定義文の入力データか、自分より前に実行された記述文の出力データの中にある。

□

上の規則の①②は、複文が未定義文の入力を使って、未定義文の出力を与えることを保証している。③は、出力の値が一意に定まることを保証するための条件である。④は、文が実行される前に、必要なデータが用意されねばならないことを述べている。また、①②の条件は、未定義文の意味が、記述文だけで表現できることを示している。すなわち、未定義文の記述文の中に未定義文が含まれていても、未定義の記述文の意味を記述した文との相互作用を考える必要がないということである。

５　プログラム論理の図式表現

５．１　順次構造の図式表現

３で定めたプログラム論理の設計手順を用いて、プログラム論理を表す等価関係の集合（以後等価集合という）を得る方法を示した。このようにして得られる等価集合は、冗長であり、また、必ずしも分かりやすいものではない。ここでは、等価集合として表されたプログラム論理を、それと等価な木構造図で表す方法を示す。

まず、プログラム論理が次のような等価集合で表されているものとする。

Ｐ≡Ｋ１；Ｋ２　　［１６］

Ｋ１≡Ｓ１　　［１７］

Ｋ２≡Ｋ３；Ｋ４　　［１８］

Ｋ３≡Ｓ２　　［１９］

Ｋ４≡Ｋ５；Ｋ６　　［２０］

Ｋ５≡Ｓ３　　［２１］

Ｋ６≡Ｓ４　　［２２］

まず、等価関係［１６］を図２（Ａ）の様に表す。これは、文Ｋ１、Ｋ２をこの順に施したものが、プログラムＰと等価であることを表している。これに［１７］の等価関係（未定義文Ｋ１が既定義文Ｓ１と等価である）を追加すると図２（Ｂ）の様になる。ここまでで、３の設計手順（１）－（６）を適用したことになり、設計の

```
P          P          P
|－K１     |－K１     |－S１
|－K２     |  |－S１   |－K２
―          |  ―       ―
           |－K２
           ―
（A）       （B）      （C）
```

図２　第１段階

第１段階が終わる。設計の第２段階以降は、（７）の手順を適用することになる。未定義文Ｋ２を新しい未定義文Ｋ３、Ｋ４で表し、さらに、Ｋ３を既定義文Ｓ２と等価であると定義する。これが設計の第２段階で、等価関係［１８］［１９］で表される。設計の第２段階が終わると、図３の木構造図が得られる。設計の第３段階は、Ｋ４の詳細化である。これにより、等価関係［２０］［２１］［２２］が追加され、木構造表現にすると図４となる。この例では、これで設計が終了する。もし、詳細化されていない未定義文があれば、すべての未定義文が詳細化されるまで、同様の手順で詳細化を続ける。

```
P
|－K１
|  |－S１
|  ―
|－K２
|  |－K３
|  |  |－S２
|  |  ―
|  |－K４
|  ―
●
```

図３　第２段階

```
P
|－K１
|  |－S１
|  ―
|－K２
|  |－K３
|  |  |－S２
|  |  ―
|  |－K４
|  |  |－K５
|  |  |  |－S３
|  |  |  ―
|  |  |－K６
|  |  |  |－S４
|  |  |  ―
|  |  ―
|  ―
●
```

図４　第３段階

仕様の冗長さをなくすために、図２（Ｂ）の表現を図２（Ｃ）の様に表すこと許される。

選択構造を表す未定義文の等価関係［７］を木構造表現したものが図５である。また、反復構造を表す未定義文の等価関係［１２］を木構造表現したものが図６である。再帰操作Ｒは、未定義文Ｌ［Ｃ］の終端を表す記号＜―＞のところで起こると解釈している。

```
|－K０［C］
|  |
|  |－IF［C］
|  |  |
|  |  T
|  |  |
|  |  F
|  |  ―
|  ―
```

図５　選択構造

```
|－L［C］
|  |
|  |－W［C］
|  |  |
|  |  K
|  |  ―
|  ―
```

図６　反復構造

６　仕様の記述

３で述べた手順により、階乗を求める計算のプログラム論理図を記述する。第一ステップとして、プログラム「階乗の計算」を未定義文「初期化」と「階乗を求める」で表す（図７）。「初期化」は、「階乗の計算」の入力「Ｎ」を受け取り、「階乗」と「数」を出力している。この「階乗」と「数」が「階乗を求める」の入力となり「階乗」が出力される。「階乗を求める」の出力のほうの「階乗」が、「階乗の計算」の出力となる。

```
◎階乗の計算（Ｎ）→（階乗）
|－□初期化（Ｎ）→（階乗、数）
|－□階乗を求める（階乗、数）→（階乗）
●
```

図７　階乗の計算（第１ステップ）

第２ステップとして、未定義文「初期化」を２つの代入式で表し（図８）、これを図７に組み込む。代入式は既定義文なのでこれ以上分割は行わない。

```
|－□初期化（Ｎ）→（階乗、数）
|  |－・階乗＝１
|  |－・数＝Ｎ
|  ―
```

図８　階乗の計算（第２ステップ）

第３ステップで、未定義文「階乗を求める」の実現を記述し（図９）、これを図７に組み込む。「階乗を求める」は前判定反復条件文を含む未定義文である。

```
|－□階乗を求める（階乗、数）→（階乗）
|  |－○前判定：数＞０
|  |  |－□積の計算（階乗、数）→（階乗）
|  |  ―
|  ―
```

図９　階乗の計算（第３ステップ）

第４ステップとして、「階乗を求める」の中の未定義文「積の計算」を２つの代入式で表し（図１０）、階乗を求めるの中に組み込む。このようにして得られたプロ

グラム論理図が図１１である。

これで、プログラム論理図の末端の文がすべて既定義文で表現されたので、プログラム論理図は完成したことになる。図１１がデータの受渡し規則を満足していることを確かめるのは容易である。

```
|一口積の計算（階乗、数）→（階乗）
|   |一・階乗＝階乗＊数
|   |一・数＝数－１
|   一
```

図１０　階乗の計算（第４ステップ）

```
◎階乗の計算（Ｎ）→（階乗）
|一口初期化（Ｎ）→（階乗、数）
|   |一・階乗＝１
|   |一・数＝Ｎ
|   一
|一口階乗を求める（階乗、数）→（階乗）
|   |一〇前判定：数＞０
|   |   |一口積の計算（階乗、数）→（階乗）
|   |   |   |一・階乗＝階乗＊数
|   |   |   |一・数＝数－１
|   |   |   一
|   |   一
|   一
●
```

図１１　階乗の計算（完成図）

７　おわりに

本手法により得られるプログラム論理図は、上に示したように厳密に定義された形式性を有している。また、基本文の意味が一意に定まれば、このプログラム論理図の意味も一意に定まる。ここでは、基本文の意味をプログラム言語で与えている。こうすることにより、プログラム論理図の意味を一意に定め、プログラム論理図からソースプログラムを生成することを可能にした。このソースプログラムの自動生成システムについては、別の機会に報告する予定である。また、プログラム論理図の矛盾を検出する設計レビューシステムを開発したが、これも機会を改めて報告する予定である。

ここで述べたプログラム論理図は、基礎理論に関する知識がなくても３で述べた設計手順により容易に作成することが出来る。

ＨＯＳのプログラム論理図との比較を、図１（ａ）と図１１、図１（ｂ）と図１２に示す。

```
|一・Ｙ＝Ａ＊Ｘ＊＊２＋Ｂ＊Ｘ＋Ｃ
```

図１２　ＡＸ２＋ＢＸ＋Ｃ　の計算

この２つの例は、本手法がＨＯＳと同じ内容をはるかに簡潔に記述できることを示している。この結果、記述した内容が理解し易くなっている。また、記述に要する時間も１／３以下に短縮できた。さらに、本手法のプログラム論理図は、日本語ワープロや日本語エディタだけで作成することが出来るのも特長である。

以上の結果、４つの設計目標をほぼ達成できたものと考えている。

本手法の中では、データ型が重要な役割を果たしている。また、二項選択や前判定反復以外の制御構造も記述できる。これらの点については、稿を改めて報告したいと考えている。

参考文献

１）野木：要求定義技術の動向、情報処理、ＶＯｌ．２０、Ｎｏ．６、ｐｐ．４８７－４９３（１９７９）

２）大野、阿草：要求仕様の記述と検証、情報処理、Ｖｏｌ．２０、Ｎｏ．６、ｐｐ．４９５－５０３（１９７９）

３）大場、金戸：代数的仕様の実用プログラムへの適用に関する考察、情報処理学会第４０回全国大会講演論文集、ｐｐ．１０２９－１０３０（平成２年前期）

４）深澤：形式仕様記述言語とその支援ツール、Ｖｏｌ．３０、Ｎｏ．４、ＰＰ．３７３－３７９（１９８９）

５）大蒔、二木：形式仕様記述言語ＬＯＴＯＳの試用経験、情報処理、Ｖｏｌ．３１、Ｎｏ．１０、ｐｐ１４００－１４１３（１９９０）

６）MARGARET HAMILTON and SAYDEAN ZELDIN: Higher Order Software - A Methodorogy for Defining Software, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol.SE-2, No.1, 9-32(1976)

７）M. Hamilton, S.Zeldin:The Relationship Between Design and Verification, The Journal of Systems and Software, Vol.1, pp.29-56(1979)

８）Ｊａｍｅｓ　Ｍａｒｔｉｎ　（成田光彰　訳）：管理職のための情報戦略、日経マグロウヒル、ｐｐ．１７７－２００

９）大場、金戸：抽象データ型に基づくプログラム設計［１］、情報処理学会第４１回全国大会講演論文集、ｐｐ．５－１８５　－　５－１８６（平成２年後期）

１０）大場　他：抽象データ型に基づくプログラム設計［２］、情報処理学会第４０回全国大会講演論文集、ｐｐ．５－１８７　－　５－１８８（平成２年後期）