Mem. Fac. Sci. Eng. Shimane Univ.
Series A
33, pp. 35-50 (1999)

代数によるプログラム構造の表現

佐藤　匡正
数理・システム情報学科　総合理工学部　島根大学

A Formal Expression of Program Structures with Algebra

Satou TADAMASA

Abstract

While studying properties of programs in designing and reviewing processes, operations such as transformation or development would be done. There is an approch of formalizing the operations with an algebra, such as the regular expression (RE). In this case the strategic and technical viewpoints would be important to develop it as methods. In the strategic one it could process the part of program semantics, while it could reduce restrictions to apply an algebra in the technical one.

This paper presents two proposals to do so. Firstly, an idea to seperate a program into two parts; mechanisms and a structure, to apply an algebra to the structure alone. A structue of a program is the nondeterministic maximum possible range, while the mechanisms could make constraint to it to be deterministic one to be desired. Secondly it is an extension of RE by adding a break for iterations, as that RE may be applied to all kinds ofiterations. It points out that the break enables program structures with continuous iteration structures in ω-RE to be developed into pre-tested or post-tested ones in Kleene closures, along with that RE can be applied to breaks in pre-tested or post-tested iterations.

梗　　概

プログラム処理の設計や審査において，処理の変形や展開といった操作が行われるが，この操作を代数（正規表現）によって形式化して扱う考えがある．この場合，方法論として実現するには基本的な考えの確立と技術的な裏付けが大切である．基本的な考えとしては，プログラムの意味を反映した部分が代数で扱えるようにすることであるし，技術上は，継続繰返しや打切りが扱えないなどの正規表現の制約を解消することである．

本論文では，このための工夫として二つの提案を行う．一つは基本的な考えで，プログラムを構造と機構に分け，構造の方に代数を適用しようとするものである．「構造」はプログラムのもつ動作の最大範囲であり，機構はこの「構造」に制約を与えて目的とするプログラム動作を規定する．もう一つは技術面で，繰返しの打切りを形式化に正規表現の拡張である．打切りの形式化によって，ω-正規表現の無限列で表したプログラム継続繰返し構造がクリーネの閉包の正規表現の前判定と後判定の繰返し構造に展開できることを指摘するとともに，正規表現の繰返し構造における打切りについても正規表現で扱えることを示す．

1. 序 論

プログラム設計や設計審査においては，流れ図などに示された処理をチェックしたり，評価したりする．こういった過程では，図を変形して様々な観点から観察したりその結果を人に説明することも少なくない．図には沢山の情報が盛り込まれているのは長所の一つである．しかし，このように図を変形するような作業では，着目している以外の情報についても変形せねばならないので手間もかかるし，複雑なため，途中に変形に伴う誤りが混入するということもある．また，複数のページに亙ると全貌が見通し難く誤りやすいとか類似性や同等性を見過すことも経験する．状況によっては図の一部の情報のみを抽象度高く表現すると有効なこともある．この一つの方法として図の定式化がある．これまでプログラムの定式化といえば，プログラム理論として知られている[2].

プログラムの動作を定式化し，数学によって正当性を与えるのが主目的であった．論理の正当性や停止性などの点からプログラム動作の正当性を保証する事を目的として述語論理を始めとする様々な試みが行なわれて来た．この方法では，数学的な取り扱いは複雑である．図の変形といった用途であれば，簡便に扱えた方がよい．

このための方法として，処理の構成方法を定式化することが考えられる．これはプログラム理論では，流れ図プログラムといわれる．流れ図プログラムは構造化されていればよく知られているように，連接，選択，繰り返しの三つの制御構造の組み合わせによって規定されている．そこで，これらの制御構造を代数の積と和の演算子に対応づければ，代数式として表現できる．

ところが，これには二つの問題がある．一つは，プログラムの脈絡であり，他は繰返しの打切りや継続の存在である．まず，脈絡の問題であるが，流れ図プログラムに対する代数式の式変形がプログラムでは意味を成さないことがある．連なりには繰り返しや選択の処理における条件記述とこれに条件を与えている処理とが動作決定のために意味上で結合することがある．この場合に，分配律を適用すると，条件の付与と選択という因果関係が逆転してしまうことがあり，これではプログラムとしては成立しない．

このような因果関係を切るための措置として，流れと意味を分ける考えがある．プログラム理論では，流れ図プログラムを「プログラム・スキーマ（以下，単にスキーマという）」と「解釈」に分けて意味の分離を図っている．流れ図プログラムの族としてスキーマを考え，これに対して変数の領域，関数，述語からなる「解釈」が与えられて流れ図プログラムになるものとしている．こうすると，解釈の方に意味が含まれスキーマの方には意味は含まれなくなるので定式的に扱える．

しかし，このスキーマは流れの制御の骨組みを表しているだけでプログラムの意味は保持していない．プログラム処理を反映しているわけではないし，分離し切れない流れの制御に関する意味も存在する．例えば，プログラミングでよく使われるスイッチやフラグなどは，本来は流れの制御の成分であるが，上の考えではこれらは取り除けず，スキーマの一部を成すことになる．したがって，スキーマの扱いはプログラム上の意味はもたない．

スキーマの扱いに意味をもたせるには，スキーマを，「解法としての意味を保存するが因

果関係は除く」ように分解すればよい．この一つとして，プログラムの成分を構造と機構に分ける，という着想を得た．機構は流れを制御する成分である．これは，選択や繰り返しの条件付けの部分だけでなく，ここに条件付けを与えるための要素処理をも含む，一つの機構を形作る部分である．構造はプログラムが行うべき本来の処理であり，機構によって生成された要素処理の列から成る．つまり，流れ図プログラムは，要素処理の列を要素とする集合が構造として定着され，この集合の要素が機構によって部分化されたものと解釈される．この機構によって部分化された集合がプログラムとしての動作を与えている．この構造には，プログラムとして実施すべき本来の要素処理の連なりのみが含まれ，流れの制御に関する意味は含まれていないので，代数式として扱えることになる．

もう一つの問題は，繰返しについてである．

実際にプログラムを観察すると，無限に繰返す（継続繰返しという）か，その途中に打切りをもつものに出会うことがある．継続繰返しは，オペレーティング・システムのコマンド・プロセッサや，トランザクション処理における電文の受け取り，線形再帰呼び出しなどがあげられる．また，プログラミング技法として，意図して継続繰返しを構成することがある．この繰返しは，繰返しの任意の箇所で打切れ便利なので実務上はよく使われる[5]．しかし，この継続繰り返しの場合，クリーネの＋と＊の閉包による正規表現では扱えない．ω-正規表現[3]を適用する必要がある．

ところが，この意図的な継続繰返しでは，この「継続」は見掛けである．実際には繰返しの打切りを前提として，繰返しのどこかで打切り条件を与えていることが多い．この様に考えると，見掛けの継告繰り返しは，クリーネの閉包による通常の正規表現に変換できそうである．一方，通常の繰返しについても打切りをもつ．これは，正規の終了の他に複数の出口をもつことであり，2次出口とも呼ばれる．この打切りの最も身近な例は，誤り処理（recovery）や例外処理である．入力の規定値チェックやシステム呼び（system call）のハンドルのチェックなどにおいて条件付けられ打切りが作られる．こうした打切りをもつ処理は，正規に終了する繰返しに対しても存在するのだから，これらも含めて一般化して形式化できる様な方法が考え出せれば効果的である．

本稿では，プログラムを代数で表現するために必要な二つの方法；脈絡を取り除く方法としてプログラムから「構造」を取り出してこれを代数で表す方法，および打切りや継続をもつ繰り返しを正規表現で表す方法について論ずる．

2. プログラムの構造

2.1 プログラム代数化の問題点

(1) プログラムと代数

逐次プログラムは，**JIS-X0128** のプログラム構成要素の表現方法[4]を用いて記述できる．この方法として図による多様な表現方法（図法）があるが，ここでは HCP 図法に着目する．この図法では，その基本的な構成方法として，階層，および基本制御構造（連接，選択，繰返し）をもっている．詳細な方法については後で論ずることにして今は基本のみに着

目する．従って，プログラムを「この構成方法による演算を要素処理に施して得られるもの」と捉えれば，HCP プログラムは代数で表すことが出来る．例えば，階層を「(,)」で，連接を「・(積)」で，選択を「＋ (和)」で，繰返しを「* (クリーネの＊閉包)」で表せば，プログラムは代数式となる．

(2) 問題点

上で繰返しは連接と選択に展開できる．したがって，構成方法は連接と選択が基本であり，HCP プログラムは和と積の代数式によって表わせる．ところが，この式をこのまま変形すると分配律の適用で，プログラムとしての意味を成さなくなることがある．この理由は，選択に含まれる要素処理とこれに条件を与えている要素処理とが脈絡を構成することのあるためである．

いま，要素処理 a, b をもつプログラム式 $a\cdot(b+c)$ を想定する．この式に分配律を適用すると，$(a\cdot b+a\cdot c)$ となる．この変形は，プログラム上の意味に考慮すると成立するとは限らない．これは，式の非決定性のためである．

変形前の $a\cdot(b+c)$ において，$(b+c)$ で b, c のどちらかを選択する条件が，基本要素 a によって与えられているものとする．この場合，( ) と，これに条件を与えている a には意味上のつながりがある．プログラムとして意味をもつ順序は，a の次に ( ) が続く $a\cdot(b+c)$ である．a と ( ) の順序の逆転した $(a\cdot b+a\cdot c)$ では意味をなさない．

この様子を例示する．

二次方程式の根を求めるプログラムを図１に示す．図で要素処理を次の様に決める．

P ；2 次方程式の根を求める，
l ；パラメータを得る，
m ；根を求め表示する，
m_1 ；実根を求め表示する，
m_2 ；重根を求め表示する，
m_3 ；虚根を求め表示する，
n ；「$D=b^2-4\times\mathrm{a}\times\mathrm{c}$」を計算する，
α ；D によって m_1 を選択する．

プログラム P は，

$$P=l\cdot n\cdot\alpha\cdot(m_1+m_2+m_3)$$

である．展開して次が得られる．

$$P'=(l\cdot n\cdot\alpha\cdot m_1+l\cdot n\cdot\alpha\cdot m_2+l\cdot n\cdot\alpha\cdot m_3)$$

P' の意味を考える．α は分岐条件を与えているのであるから ( ) と対で，かつ，$\alpha\cdot(\cdots)$ の順序で意味をもつ．P はこの形式であるが，式 P' では，α が ( ) 内に移った，$(\cdots\alpha\cdots)$ の形式である．この順序では選択した後にその選択条件を与えると解釈される．この解釈はプログラムの因果律には反しており，プログラム上の意味を失う．これは代数式

の変形が出来ない．適用できるようにするには何らかの工夫が必要である．

2.2 機構と構造

(1) 適用の考え方

上の不都合は，要素処理と選択が脈絡をもつ，つまり「条件（値）を介して結合しているため」である．この解決には次の二案が考えられる．

① そのまま代数式にする．式の変形結果については吟味して妥当性を確かめる．

② プログラムから結合をもたない部分を取り出して代数で扱う

①案は，変形結果がプログラム処理として意味をもつか否かを吟味するもので適用は容易であるが，範囲は限定される．②案は，脈絡を切る考えである．ただし，取り出した部分がプログラム上の意味をもてば代数で取り扱う価値が生まれる．

このために，プログラムが，「機構」と「構造」の二つの部分からなり，プログラムの意味を反映すべく構造に機構を働かせてこれを部分化したものと解釈する．機構は，各々の選択において，それに含まれる要素を選択する働きである．この仕掛けによって，構造に含まれる要素処理の組み合わせ方に一定の制約が与えられる．

機構の例として，スイッチやフラグがある．機構に含まれるものは，初期値設定部分，オン/オフの判断部分，およびオン/オフの操作部分である．この仕組みによって実行が制御され，その結果，構造が部分化される．一方，構造はプログラムからこの機構を除いた部分である．基本的には，そのプログラムが本来果たすべき役割の大枠であり，そのプログラムの入力と出力を結び付ける要素処理の連なりである．

(2) プログラムでの意味

上で述べた構造と機構を言い換えると次のようになる．すなわち，プログラムを構造と機構に分けて考えたとき，構造は要素処理の連なりの集合であり，このプログラム処理が生成できる最大の集合を表している．機構は，この集合を部分化するための仕掛けである．

機構と構造を，図1の二次方程式の解法を例として示す．プログラムは次の要素処理の連なりの集合である．

$$\{l\cdot n\cdot\alpha\cdot m_1,\ l\cdot n\cdot\alpha\cdot m_2,\ l\cdot n\cdot\alpha\cdot m_3\}$$

機構は，要素処理 l, α, n によって構成された $l\cdot n\cdot\alpha$ である．集合のいずれかの要素を選択する仕掛けが形成される．構造は $\{l\cdot m_1, l\cdot m_2, l\cdot m_3\}$ である．また，プログラムの入力は l で生じ，出力は $m_i(i=1, 2, 3)$ で生ずる．

(3) 構造における脈絡

脈絡は機構に含まれるので構造には，機構によって取り除かれる筈の連なりが含まれることがある．例えば，$a+\sqrt{b}$ の値を求めるプログラム（図2）は，機構によって生成される連なりは次の2要素の集合である．

$$\{h\cdot\alpha\cdot l_1\cdot m\cdot\alpha\cdot n_1\cdot k,\ \alpha\cdot l_2\cdot m\cdot\alpha\cdot n_2\}$$

しかし，構造は，4要素の集合となる．

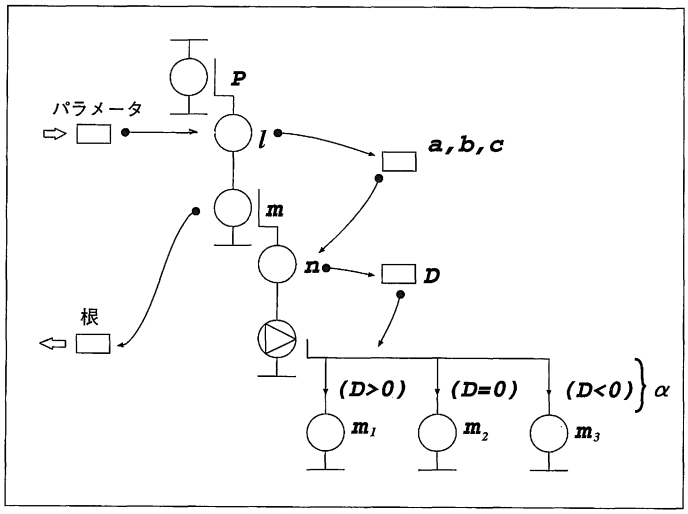


図1 二次方程式の根を表示する

$$\{l_1 \cdot m \cdot n_1 \cdot k,\ l_1 \cdot m \cdot n_2,\ l_2 \cdot m \cdot n_1 \cdot k,\ l_2 \cdot m \cdot n_2\}$$

構造には，プログラムとしては存在しない二つの連なり $l_1 \cdot m \cdot n_2$, $l_2 \cdot m \cdot n_1 \cdot k$ がある．

2.3 機構と構造の分解

(1) 分解の方法

構造と機構とを区別するには機構の方に着目する方が考えやすい．基本的には，プログラムから機構を取り除くことによって構造が取り出される．ただし，プログラムには機構と構造が混在していて互いに共用している部分がある．この部分は構造の一部として残さねばならない．

プログラムで機構を見つけるにはその構成に着目すれば容易である．機構は条件決定用の情報参照とその情報設定の連なりから構成されているので繰り返しや選択で用いられている変数を鍵としてその変数に情報を設定している要素処理を見つければよい．構造を得るには解法からこの機構を取り除けばよい．但し，機構と構造が要素処理を共用することがある．この場合は単純に機構を取り除くと構造の一部が欠けてしまうので共用部分が構造から除外されないような配慮が必要となる．

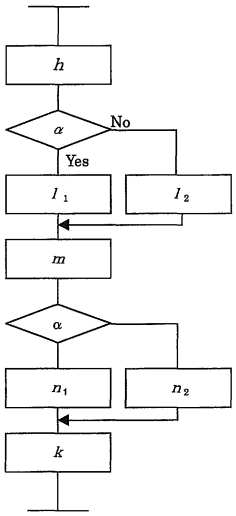
次に示す措置によって構造が取り出される．

措置1 条件付けの削除

解法の繰り返しおよび選択の要素処理における条件記述を削除する．

措置2 機構用変数の確定

上の条件記述における変数のうちで条件を与えている変数を識別する．この変数を機



h ： a、bを得る
l_1 ： $w \leftarrow b$
l_2 ： $w \leftarrow -b$
m ： $t \leftarrow \sqrt{w}$
n_1 ： $d \leftarrow a + t$
n_2 ： $a +$「i」t
k ： dを出力する
α ： $b > 0$を判断して分岐する

図2　$a+\sqrt{b}$ の値を求める

構用変数ということにする．機構用変数は，プログラムの全ての要素処理においてその変数について情報は設定されるが，繰返しや選択を除いては参照されていないものである．

措置3　機構の削除

この機構用変数に対して値を与えている操作，もしくは参照している操作をもつ要素処理を削除する．

このように機構と構造に分割することによって，積の分配律における不都合は機構の方に閉じ込められるので，構造は代数式で表現できるようになる．

この構造を表す代数式では，機構についての言及ができないので，プログラム全体の形式化はできていない．従って，これまでに「プログラムの理論化」として行なわれてきたようなプログラムの正当性について論ずること[2]はできない．しかし，プログラムについて全体の構成などを包括的に議論するには便利である．

(2) 例示

① 選択機構

図1の二次方程式の解法を例として機構と構造を示す.

本例での機構は選択の条件決定で使用されている. この条件記述の変数 D に着目する. D の値を設定している n, n の a, b, c に値を与えている l が確定する. つまり, 機構は要素処理である $l, n,$ および α から成る. α は選択要素処理である.

構造は措置1, 2から得られる. 措置1から選択要素処理 α が除かれ選択での変数 D が定まる. n によって D は値が設定される. この D は, m_1, m_2, m_3 で参照されるから n は除かれない (措置2). こうして構造が確定する.

② カウンタとフラグの機構

カウンタ機構の例として, ブロッキング処理を, フラグ機構の例として重複チェック処理をとりあげる.

図3に文字列を一定の長さに切り出すプログラムを示す. 図3には繰返しと選択がある. 選択 n の変数はカウンタであり, これについての c_1, c_2, および条件「(カウンタ>10)/(それ以外)」である. 繰返しの変数は「字」で, これに係わる要素処理は g と条

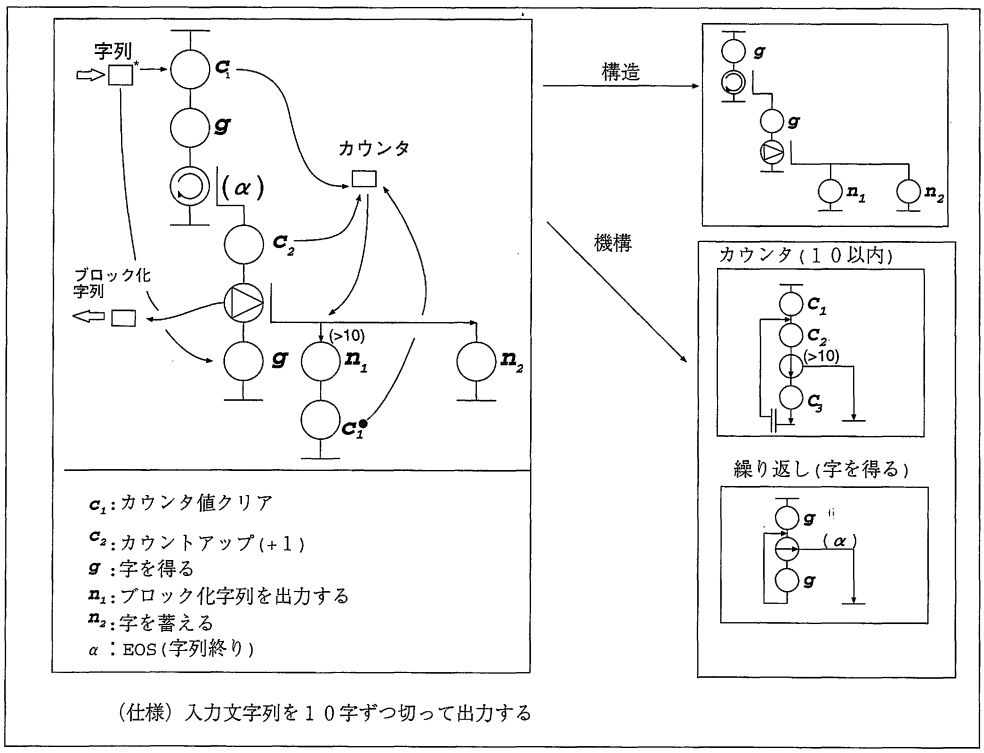


図3 文字列のブロッキング

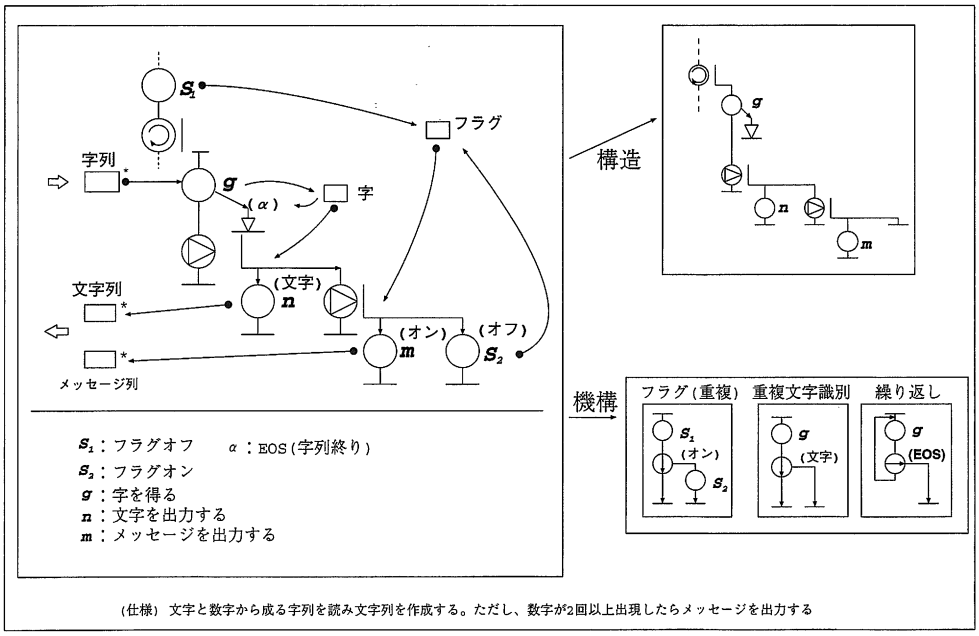


図4　重複出現のチェック

件「**EOS** 字列終わり)」である．gは，nに対する措置2により構造として残る．

図4には文字列中に同一の文字が重複して出現したか否かを調べるプログラムである．図では機構は3個存在している．繰り返しが1個，選択が2個（フラグと文字識別）である．繰り返しの変数は「字」であり，選択での機構用変数は「フラグ変数」と「字」である．これらの変数に係わる要素処理から構造が確定する．いま，nに対して措置2が適用され，gは削除されずに構造の一部として残る．

3. 代　　数

HCP プログラムの構造の記述ができるように代数の演算子およびフルファベットを拡張する．

3.1 基本項目

いま，アルファベット {ε, 英字} をもつ正規表現を考える．εは空元を表す．演算子は {+, ・, *, $^+$} とする．式として（　）を用いる．

3.2 閉包の打切り

上の正規表現に，アルファベットξを，および演算子 {$\infty, \omega, \rfloor$} を加え拡張する．これ

らの定義を次に与える．

なお，以下では，積の演算子を省略するために代数の単項式を大小英字（ギリシャ文字を含む）の１字で表すことにする．但し，これらの使い分けは，単に式の見やすさを狙いとしたもので，特別な意味をもつものではない．また，・演算子は煩雑なので誤解のない限りは省略する．

(1) ∞閉包[3]

有限アルファベット上の有限語$f(f\subseteq A^*)$に対してω閉包を次で定義する．

$$f^{\omega}:=\{\alpha\in A^{\omega}\mid \alpha=w_0w_1\cdots,\ \text{全ての}\ i\geqq 0\ \text{に対して}\ w_i\in f\}$$

このとき，∞閉包は次とする．

$$f^{\infty}:=f^{\infty}\cup f^*$$

(2) 打切りの定義

打切りは無限の連なりを有限にするための演算子である．この定義を考える．いま，打切りを記号」で表すものとする．この演算子は∞閉包内でのみ意味をもち，アルファベットの直後に置かれるものとする．

いま，正規表現fに打切り」が含まれているものとする．ただし，∞閉包は含まれないものとする．fは一般に連接項の和形式に展開できるから，展開形が打切りを含むか含まないかによって分けることができ，次の様に書ける．

$$f=f_R+f_E$$

ここで，f_Rは打切りを含まない有限個の積の和形式であり，f_Eは一つだけの打切りを含む有限個の項の和形式である．f_Eを確定させる．いま，f_Eのi番目の積の項e_iを打切りの位置によって３分割する．打切り」の直前の単項をb_iとし，先頭からb_i項の直前までの積の項をa_iとし，b_i項の次から末尾までの積の項をc_iとすればe_iは次となる．

$$e_i=a_ib_i\rfloor c_i$$

よって，N個の項をもつf_Eは次となる．

$$f_E=a_1b_1\rfloor c_1+a_2b_2\rfloor c_2+\cdots+a_Nb_N\rfloor c_N$$

このとき，打切りの演算は次の式１で定義される．

$$\begin{aligned}f^{\infty}&=(f_R+f_E)^{\infty}\\&=(\mathrm{f_R}+\mathrm{a_1}b_1\rfloor c_1+\mathrm{a_2}b_2\rfloor c_2+\cdots+\mathrm{a_N}b_N\rfloor c_N)^{\infty}\\&=f_R{}^{\omega}+f_R{}^*(a_1b_1+\mathrm{a_2}b_2+\cdots+a_Nb_N)\end{aligned}\qquad(1)$$

(3) ∞閉包の展開形

まず，アルファベットξを定義する．ξは仮想元で，実在しない仮想の列を意味する．a, bを任意の英字または空元εとしたとき，次の性質をもつ．

$$\xi+a=a+\xi=a$$
$$\xi a=\xi,\ a\xi=\xi,$$
$$a\xi b=a(\xi b)=(a\xi)b=\xi$$
$$\xi^{+}=\xi,\ \xi^{*}=\varepsilon+\xi^{+}=\varepsilon+\xi$$

式(1)で，$f_R^{\infty}=\xi$ とすると次になる．

$$f^{\infty}=f_R^{*}(a_1b_1+a_2b_2+\cdots+a_Nb_N) \tag{2}$$

式2より打切りをもつ∞閉包は正規表現である．

(4) ∞閉包の入れ子

打切りをもつ∞閉包の入れ子形式は，次の定理によって正規表現に展開できる．

定理：∞閉包の有限回の入れ子は正規表現である．

証明：n 重入れ子の最も内側の正規表現を f_0 とする．f_0 が打切りをもたないとすると，$f_0^{\infty}=\xi$ である．また f_0 に打切りを含むとすると，式(1)の定義によって正規表現に展開できる．次に，f_0 の∞閉包による正規表現を f_1 とすると，この∞閉包も正規表現に展開できる．この手続きを n（有限回）繰り返せば，∞閉包は f_n の正規表現に展開できる．（証明終り）

(5) ＊閉包および＋閉包との関係

∞閉包と，＊閉包および＋閉包との関係について考察する．

いま，次を考える．

$$p_{pre}=((\varepsilon+\varepsilon\rfloor)a)^{\infty}$$

式(2)を適用すると次のように展開できる．

$$=(\varepsilon a+\varepsilon\rfloor a)^{\infty}$$
$$=(\varepsilon a)^{\omega}+(\varepsilon a)^{*}\varepsilon=a^{*}\varepsilon=a^{*} \tag{3}$$

同様に，

$$p_{post}=(a(\varepsilon+\varepsilon\rfloor))^{\infty}=(a\varepsilon+a\varepsilon\rfloor)^{\infty}$$
$$=(a\varepsilon)^{\infty}+(a\varepsilon)^{*}a\varepsilon$$
$$=a^{*}a=a^{+} \tag{4}$$

この結果から，＊閉包および＋閉包は∞閉包に対して，打切りをそれぞれ最前項に，末尾項にもつ式に帰着できることがわかる．これは∞閉包の特別な形式が＊閉包および＋閉包であると解釈できる．

4. プログラム構造との対応

4.1 継続繰返しと前判定および後判定の繰返しの関係

∞閉包のプログラム構造との対応を考察する．基本的には，**HCP** 図法で表した構造について，要素処理をアルファベットで，制御を演算子にそれぞれ対応づける．連接は・で，選

択は＋で表す．繰返しについては，前判定（DO–WHILE; pre-tested），後判定（REPEAT–UNTIL; post-tested），継続（continuous）の三種類が規定されている[4]．この内，**HCP** 図法では継続繰返しを基本としている．そこで，継続繰返しを∞閉包で表したときの前判定および後判定がどの様に対応するかを考察する．

(1) 前判定繰返し

前判定繰返し処理は，継続繰返しにおいて繰返し部分の先頭に打切りを設けたものであるから，見掛けの∞閉包で扱える．この処理から「構造」を取り出すと，a とは脈絡をもたない条件付けによって繰返しを打ち切る図5(1)となる．図の三角の重なりは打切りである．この構造として次が得られる．

$$\begin{aligned} d&=((\varepsilon+\varepsilon\rfloor)a)^{\infty} \\ &=(\varepsilon a))^{\omega}+(\varepsilon a)^{*}\varepsilon=a^{*}\varepsilon=a^{*} \end{aligned} \tag{5}$$

よって，式(3)から，

$$d=p_{pre} \text{ である．}$$

つまり，前判定の構造は＊閉包で表わせ，継続繰り返し構造においてその先頭に打切りを設けた構造と同等であると言える．

(2) 後判定繰返し

後判定は繰返し部分の末尾に終了条件を設けた表記方法である．上と同様に，図5(2)の構造が得られる．これは次式となる．

$$r=(a(\varepsilon+\varepsilon\rfloor))^{\infty}$$

これは，式(4)により，

$$r=p_{post} \text{ である．}$$

従って，この後判定の構造は＋閉包で表せ，継続繰り返し構造においてその末尾に打切りを設けた構造と同等であると言える．

(3) 一般形

打切りをもつ継続繰返しの一般形の構造を求める．この一般形として後処理 c をもつ構造の次を考えれば十分である．

$$q=(a(b+c\rfloor)d))^{\infty}=(abd+ac\rfloor d))^{\infty}$$

式(2)より次が得られる．

$$=(abd)^{*}(ac)$$

4.2 打切りの形式化

継続繰り返しの打切りの形式化については上で述べた．次に，前判定や後判定の繰り返し

における打切りの形式化を考える．この一般形として，前判定の繰り返しにおいて後処理 c をもつ構造を考えれば十分である．

$$P=(a(b+c\rfloor)d)^*$$

式(5)より，∞閉包で表すと次となる．

$$\begin{aligned}&=((\varepsilon+\varepsilon\rfloor)(a(b+c\rfloor)d))^\infty\\&=(abd+ac\rfloor d+\varepsilon\rfloor abd+\varepsilon\rfloor ac\rfloor d)^\infty\\&\quad \varepsilon\rfloor abd=\varepsilon\rfloor,\ \varepsilon\rfloor ac\rfloor d=\varepsilon\rfloor \text{ だから}\\&=(abd)^*+(abd)^*(ac)\end{aligned}$$

上の式の第2項が打切りを表している．

後判定に関しても同様に次で表される．打切り分は第2項である．

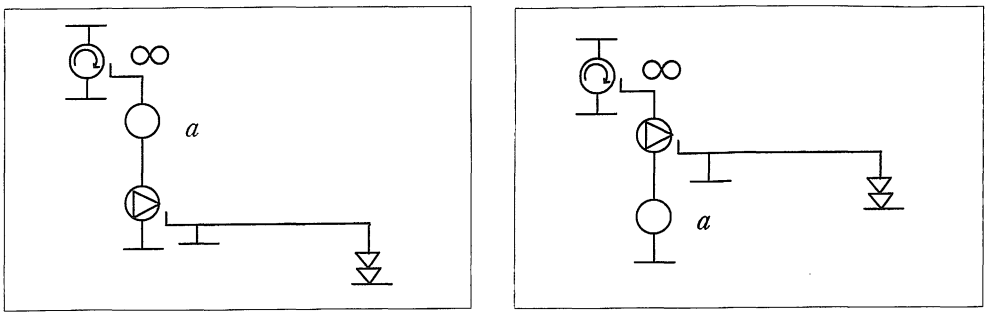
$$\begin{aligned}P&=(a(b+c\rfloor)d)^+\\&=(abd)^++(abd)^*(ac)\end{aligned}$$

4.3 打切りの例

(1) 同等処理のチェック

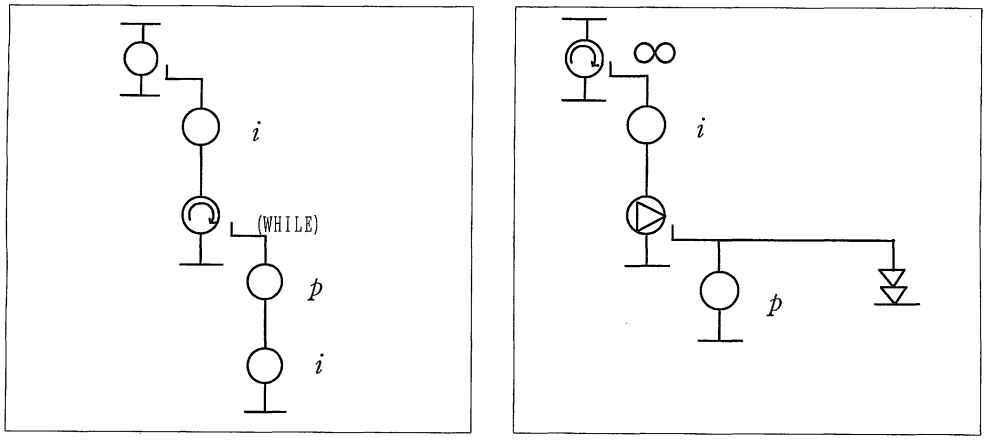
プログラムの同等性（プログラム変異体という）を構造によって確かめることが出来る．いま，文献5で同等とされているプログラムについて考える．ここでは，簡単なプログラムの同等性が図によって示されている．

この例は，鍵盤からの入力を画面に繰返して表示するもので，入力が‘入力終り’という文字列になると終了する．これらの構造を図6(1),(2)に示す．図で，i, p は要素処理で，それぞれ，「打鍵による入力」，「画面への表示」である．これらの構造が同一であることを示す．



(1) 前判定（DO-WHILE 型）　　(2) 後判定（REPEAT-UNTIL 型）

図5 継続繰返しと前・後判定繰返し



(1) 前判定の構造　　(2) 同等の無限構造

図6 無限構造の例

前判定をもつプログラム処理（図6(1)）の構造を T とする．

$$T=i(pi)^*$$

一方，継続繰返し処理（図6(2)）の構造を S とすると，

$$\begin{aligned} S&=(i(p+\varepsilon\rfloor))^\infty \\ &=(ip+i\varepsilon\rfloor)^\infty=(ip)^*i \\ &=i(pi)^* \end{aligned}$$

よって，$T=S$ である．両者は同一の構造をもつといえる．この形式化では構造についてだけの議論であり，構造の同等性でプログラムとしての動作が一致するとはいえない．しかし，この例からも分かるように処理構成についての大枠の議論は可能である．

(2) 重複出現チェック

継続繰返しを使って任意の箇所で打切りを指定するのはよく行われる．この例として，図4における重複出現チェックを考える．この構造と同等な構造を探ってみる．

いま，この構造を P_3 とする．

$$P_3=(g\cdot(\varepsilon+\varepsilon\rfloor)\cdot(n+(m+\varepsilon)))^\infty \tag{6}$$

和の積および打切りを展開する．

$$P_3=(g\cdot(n+m+\varepsilon))^*\cdot g \tag{7}$$

さて，$(x\cdot y)^*\cdot x=x\cdot(y\cdot x)^*$ であるので，

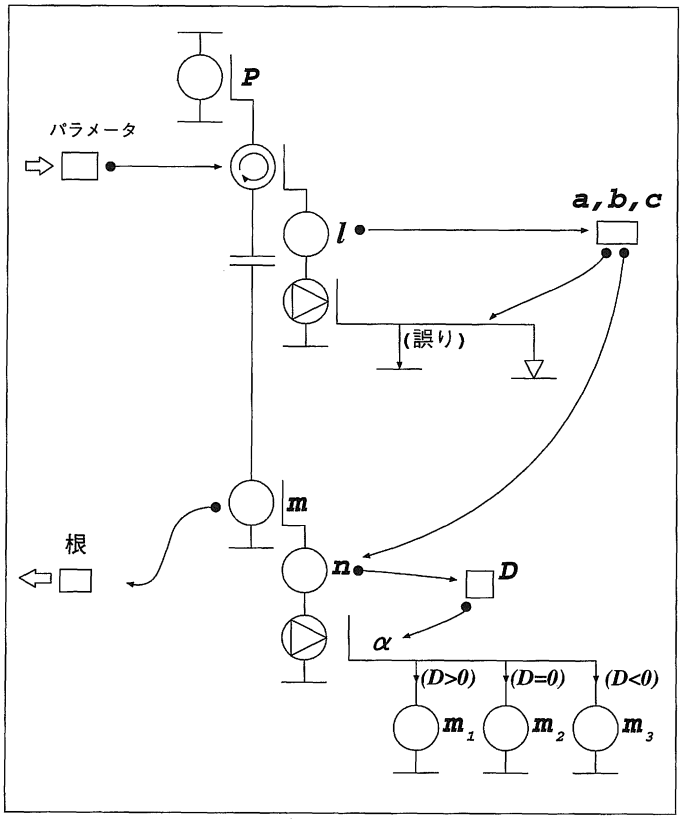


図7　誤り回復処理（図1に対して）

$$P_3=g\cdot((n+m+\varepsilon)\cdot g)^* \tag{8}$$
$$\subseteq((\varepsilon+g)\cdot(n+m+\varepsilon)\cdot g)^* \tag{9}$$

式(8)は前判定繰返しである．また，式(9)は（$\varepsilon+g$）の選択でフラグ機構を使って繰返しの初回のみgが選択されるようにすれば式(8)と同等となる．

(3) 誤り回復処理

継続繰返しは誤り回復処理に使われることが多い．いま，図1の二次方程式の解法において，パラメータのチェックを行うように変更する（図7）．パラメータに妥当性のないと正当な値の入力されるまで入力が繰り返されるものとする．この構造は次となる．

$$S=(l(\varepsilon+\varepsilon\rfloor)^\infty n(m_1+m_2+m_3))=l^+n(m_1+m_2+m_3)=l^*\cdot ln(m_1+m_2+m_3)$$

上の式のl^*は誤りによってパラメータを入力した数を表している．

4. 結　　論

HCP プログラムを代数で扱う方策として，構造と機構に部分化する方法，および繰返しの打切りを形式化する方法を提案した．

構造は，解法のもつ意味を反映した要素処理からなる部分集合であり，機構は，この集合を限定する部分集合である．また，打切りの形式化によって ω-正規表現の無限列で表したプログラム継続繰返し構造がクリーネの閉包の正規表現の構造に展開できることを指摘した．さらに，全ての繰返し構造における打切りを正規表現で扱えることを示した．これらの提案によって，プログラム構造を正規表現で扱えるようにした．

参 考 文 献

[1] 佐藤匡正：HCP 図法で記述されたプログラム解法の S 代数による定式化，情報処理　論文誌 Vol. 27 No. 6 (1986).

[2] Zohar Manna: Mathmatical Theory of Computation, McGraw-Hill Inc. (1974) 五十嵐訳，プログラムの理論　日本コンピュータ協会.

[3] Wolfgang Thomas: Automaton on Infinite Objects (Cp. 4) in HANDBOOK OFTHEORETICAL SCIENCE VOL. B, Elseveir Science Publishers B. V., 1990，山崎訳，無限の対象上のオートマトン，丸善（平成 6 年 (1994)).

[4] JIS ハンドブック　ソフトウェア編，JIS X 128-1988，プログラムの構成要素及びその表記法，日本規格協会.

[5] 佐藤匡正：プログラミング入門　～プログラム作成の考え方～，p. 66　大学教育出版 (1997) ISBN4-88730-240-1 C1055.