ソフトウェア基礎論 22－5
（１９８７．１０．１）

Ｃｏｎｔｉｎｕａｔｉｏｎ を用いたプログラム変換

長田　博泰
北海道大学情報処理教育センター

本論文は 、Continuation を用いて再帰的関数を逐次型あるいは扱い易い再帰的関数に変換する体系的方法を述べる。Continuationの方法をできる限り広範囲の変換に適用するため、とくに、Continuationの使用法をを拡張し、２個以上の変数を付加する関数の一般化を扱うことができるようにした。また、Continuationを用いた変換を一層、有効にするために有用な関数の種々の性質の利用法についても論ずる。

ここで提案する方法を二、三の例を用いて他の方法と比較検討し、Continuation による方法が有効であることを示す。

PROGRAM TRANSFORMATION USING CONTINUATIONS

Hiroyasu NAGATA
Center for Information Processing Education Center, Hokkaido University
Kita 10 Nishi 5, Kita-ku, Sapporo, 060 JAPAN

This paper presents a systematic method for transforming a recursive function into a iterative-type or tractable one using continuations. To apply to a various type of recusive functions, in particular, the use of continuations is extended to an addition of more than one variable. Also, several useful properties of functions are given for converting recursive functions with continuations.

The method is illustrated and compared with the other one of generalization by a few examples.

１．はじめに

与えられた問題が再帰的な場合、例えば、扱うデータ構造が再帰的な場合など、問題から自然に、また、容易に再帰的関数またはプログラムを作ることができる。しかも、再帰的プログラムはその意味も簡潔明瞭であるばかりでなく、その正しさを証明することも比較的容易である。再帰的プログラムはこのような利点を有するにも拘らず、一般に実行効率が悪い。

これを改善するために、従来から、再帰性の除去、あるいは逐次型再帰プログラムに変換する研究が行なわれ、また、実験的システムも作成されてきた。その手法を大別するとつぎの４つになる［１０］。すなわち、局所的簡約化、部分評価、抽象化および一般化である。ここで扱うContinuationによる方法は、このなかの一般化に属する。一般化とは、後の変換が行ない易いよう関数を置換する手法であり、このために変数の追加や関数の拡張などが用いられる。しかし、例えば、［２、３］などに見受けられるように、どのように一般化するかは発見的傾向が強く、いわば"カン"と経験によるところが大きい。自動変換システムを実現するにはこの点を解決しておく必要があり、既に［１］はこの方向に沿って、一般化、帰納法および非手続き言語の三つを併用する体系的変換手法を提案している。

本論文もプログラム変換の体系的方法の開発を目標とするが、その方法が異なる。ここでの考え方はプログラムの表示的意味論で用いられるContinuationを利用することである［７］。この方法は既に［９］が論じているが、発想自身はとくに目新しいものではなく、プログラマーが通常よく用いている方法を体系化したものと考えて良い。その考え方はつぎのとおりである。Continuationは、後に続くあるいは続くべき計算過程の表現とみなすことができる。このContinuationの閉じた式（閉式）、すなわち、自由変数を含まないλ-式を見い出し、これをある適当なデータ構造で表現することによってプログラム変換に利用する。言い換えれば、あらかじめ結果を予想し、その情報を利用して変換に役立てるという発想である。

本論文では、Continuationを利用してプログラム変換する手法を述べるとともに、とくに、次の２点について［９］の方法を拡張する。

１）Continuationを拡張し、２個以上の変数を付加する関数の一般化に適用することができるようにした。

２）Continuationを表現するデータ構造を見付け易くするため、Continuationの閉じた形の表記法を導入した。

また、ここで拡張した方法を用いることによって、再帰的プログラムが機械的に変換することができることを［１］が取り上げている例を用いて、比較検討し、Continuationによる方法が有効であることを示す。

２では、本論文を理解するために必要な事項とプログラム表記法等について述べる。３では、Continuationを用いる変換の基本的考え方とそのデータ構造を見付ける手法について説明する。４では、Continuationを拡張し、二変数以上を付け加える関数の一般化を扱う方法を提案するとともに、Continuationの適用に有効な関数の性質の利用法を検討する。５では、二、三の再帰的関数を用いて、他の方法、とくに、［１］が論じている方法と比較検討し、Continuationによる方法が自動的機械的変換システムの実現に適していることを示す。

２．準備

ここでは、本論文の展開に必要な定義、記法および言語について記述する。しかし、λ計算についてはよく知られているものと仮定する。

２．１　プログラム言語

再帰的関数の記述には、以下のようなｌｉｓｐ風の言語を用いる。

（１）空リストを［］で表わす。

（２）cons(x,y)は一番目の要素がｘ，二番目の要素がｙであるようなリストを構成する。
cons(x_1,cons(x_2,...,cons(x_{n-1},x_n)...)) を［x_1,x_2,...,x_{n-1},x_n］と記述する。

（３）hd(headを意味する)は空でないリストからその第一番目の要素を取り出す関数である。例えば、
hd(cons(x,y))=x.

（４）tl(tailを意味する)は空でないリストから一番目の要素を除いたすべての要素を取り出す関数である。例えば、tl(cons(x,y))=y.

（５）条件式を以下のように表現する。
if p_1 then e_1 else if p_2 then e_2 ... else e_n

（６）再帰的関数を以下のように記述する。
f(x_1,x_2,...,x_n) <== 条件式

２．２　Continuation

ある式または命令Ｍに対して、Ｍの評価が終わった後つぎに何をするかあるいは何が起こるべきかを指示するには、Ｍへの引数としてつぎの動作を規定するものを渡せばよい。このような次の動作を Continuation という。

再帰的関数 $f(x_1,x_2,\ldots,x_n)$に対し $fc(x_1,x_2,\ldots,x_n,\gamma)$ がつぎの関係を満たすならば、 fc を f の Continuation-passing 型と呼ぶことにする。

（＊） $fc(x_1,x_2,\ldots x_n,\gamma)=\gamma(f(x_1,x_2,\ldots,x_n))$

例えば、

```
f(x) <== if x=0 then 1 else f(x-1)+2
```

の Continuation-passing 型は以下のようになる。

```
fc(x,γ) <== if x=0 then γ(1)
                else fc(x-1,λw.γ(w+2))
```

この fc が（＊）を満足することはサブゴール帰納法［５］を用いて以下のように示すことができる。

x = 0 ならば、f の定義から

$fc(x,\gamma)=f(0,\gamma)=\gamma(1)=\gamma(f(x))$

x ≠ 0 かつ $fc(x,\lambda w.\gamma(w+2))$ の場合

$fc(x,\gamma)$

$=fc(x-1,\lambda w.\gamma(w+2))$

$=(\lambda w.\gamma(w+2))(f(x-1))$ （帰納法の仮定）

$=\gamma(f(x-1)+2)$ （β- 変換）

$=\gamma(f(x))$ （f の定義）

よって、（＊）の成立していることが証明された。

２．３　再帰関数の分類

本論文で用いる再帰関数を［８］にしたがって分類する。各型の定義は以下のとおりである。定義しようとする再帰関数の名前を関数記号と呼ぶことにする。

（１）iterative 型

関数記号が右辺の一番外側に高々１個出現する。

（２）linear 型

関数記号が右辺のある関数の内側に高々１個出現する。

（３）simple 型

再帰関数の木表現（ただし、共通部分式を簡約化した木）の葉からルートに至るどの道も出現するすべての関数記号を通過する。

２．４　記法

関数の合成を"∘"で表現する。すなわち、

$g(f(x))=(g\circ f)x=(g\circ f)(x)$.

また、$f(f(\ldots f(x)\ldots))=f^n(x)$ のように表わし "∘"を省略する。また

```
h(x) <= if p(x) then f(x) else g(x)
```

を，以下のように表現し、これを関数の直和と呼ぶことにする。

$(p\rightarrow f,g)$ または (f,g)

一般に、関数が

if $p_1(x)$ then $f_1(x)$ else if $p_2(x)$ then $f_2(x)$ .. ..else $f_n(x)$

ならば、

$(p_1\rightarrow f_1,\ldots,p_{n-1}\rightarrow f_{n-1},f_n)$または$(f_1,f_2,\ldots,f_n)$

である。関数の合成と組み合わせて

$e\circ(f,g)=(e\circ f,e\circ g)=(p\rightarrow e\circ f,e\circ g)$

$(f,g)\circ e=(f\circ e,g\circ e)=(p\rightarrow f\circ e,g\circ e)$

と定義し、

$(f,g)^1=(f,g)$

$(f,g)^{i+1}=(f,g)\circ(f,g)^i$

とする。

３．Ｃｏｎｔｉｎｕａｔｉｏｎによる変換

既に述べたように、Continuationを用いた変換はContinuationの閉じた式をあるデータ構造として表現することによって、変換の見通しを得易くすることである。ここでは、まず、Continuationによる方法の基本的戦略を説明する。ついで、この方法を再帰的関数の種々の型に適用し、その一般性と問題点を明らかにする。

３．１　基本的戦略

Continuationによるプログラム変換はつぎの四つのステップを踏む。

（１）再帰関数をContinuation-passing 型に変形する。

（２）Continuationの閉じた式を見出す。

（３）これをあるデータ構造で表現する。

（４）このデータ構造を利用してContinuation-passing 型の関数が元の再帰関数を計算するよう変換する。

以上四つのステップをつぎのMaCathyの９１関数を用いて説明しよう。

```
f(n) <== if n>100 then n-10
         else f(f(n+11))
```

これは、

$$f(n)=\begin{cases} n-10 & n>100 \\ 91 & n\leq 100 \end{cases}$$

となる関数である。

この関数f(n)を条件（＊）を満たすfc(n,γ)に変換する。そのためには、右辺のfの引き数をfcの引き数とし、fを含む式を変数で置き換えた式にContinuationを適用する。fを含まない式には、単に、γを適用する。したがって、

```
fc(n,γ) <== if n>100 then γ(n-10)
            else fc(n+11, λw.γ(f(w)))
```

これが（＊）を満足することは２節と同様に証明することができるので省略する。
つぎに、γの閉じた式を見つけるために、γと右辺のfcの$\lambda w.\gamma(f(w))$との関係を調べる。$\gamma=\lambda v.f(v)$とする。

$$\lambda w.\gamma(f(w))=\lambda w.f(f(w))$$

したがって、γの閉じた式は $\lambda w.f^{i}(w)$ である。これは、fの適用回数をカウントする変数を用いて表現することができるので、以下のように変換される。

```
f(n) <== fc(n,0)
fc(n,i) <== if n>100 then γ(n-10)
           else fc(n+11,i+1)
```

また、$\gamma(n-10)$は、

$$\gamma(n-10)=(\lambda w.f^{i}(w))(n-10)=f^{i}(n-10)$$

であるから、これを次の補助関数の導入によって計算することができる。

```
fcsend(v,i) <== if i=0 then v
               else fc(v,i-1)
```

最終的に、fcは

```
fc(n,i) <== if n>100 then fcsend(n-10)
           else fc(n+11,i+1)
```

となる。fcは逐次型の再帰関数になっているから、これを逐次型のプログラムに変換するのは容易である。

３．２　種々の型の再帰関数への適用

［８］にしたがって、再帰関数をlinear - nonlinear, simple - not simple の型に分類することとし、これらの各々に対しContinuationを用いた変換を試みる。これによって、Continuationによるプログラム変換の手法が一層明確になるばかりでなく、適用する際の問題点も明らかにすることができる。以下、各々の場合について検討しよう。

（１）ｌｉｎｅａｒ－ｓｉｍｐｌｅの場合

つぎの関数はこの型の例である。

```
f(x) <== if p(x) then a(x)
        else b(f(c(x)))
fc(x,γ) <== if p(x) then γ(a(x))
           else fc(c(x), λw.γ(b(w)))
```

Continuationの閉じた式は $b(b(\ldots b(w)\ldots))=b^{i}(w)$ となる。iをカウンタとして用いればfcをつぎのように表現することができる。

```
fc1(x,i) <== if p(x) then γ(a(x))
           else fc1(c(x),i+1)
```

$\gamma(a(x))=b^{i}(a(x))$は

```
fcsend(v,i) <== if i=0 then v
               else fcsend(b(v),i-1)
```

によって求めることができる。結局、fは以下のように変換される。

```
f(x) <== fc1(x,0)
fc1(x,i) <== if p(x) then fcsend(a(x),i)
           else fc1(c(x),i+1)
```

また、定義しようとする関数が複数の枝、すなわち、条件ごとに出現する場合も、全く同じように扱うことができる。その例を以下に示す。

```
f(x) <== if p(x) then e(x)
        else if q(x) then a(f(b(x)))
             else c(f(d(x)))
fc(x,γ) <== if p(x) then γ(e(x)) else
           if q(x) then fc(b(x),λw.γ(a(w)))
                   else fc(d(x),λw.γ(c(w)))
```

この場合，Continuationの閉じた式は、関数の直和を用いれば容易に以下のように表現するこができる。

$$\gamma w.(a,c)^{i}(w)$$

したがって、上述の方法と同じようにカウンタを用いて変換することができる。

```
fc1(x,i) <== if p(x) then fcsend(e(x),i)
           else if q(x) then fc1(b(x),i+1)
                else fc1(d(x),i+1)
fcsend(v,i) <== if i=0 then v else
              if q(x) then fcsend(a(v),i-1)
                      else fcsend(c(v),i-1)
```

（２）ｌｉｎｅａｒ－ｎｏｔ　ｓｉｍｐｌｅ

この型の最も簡単な例を以下に示す。

```
f(x) <== if p(x) then a(x)
        else b(f(c(x)),d(x))
```

そのContinuation-passing 型は

```
f(x,γ) <== if p(x) then γ(a(x))
          else fc(c(x),λw.γ(b(w,d(x))))
```

である。γの閉じた式はつぎのようになる。

$\gamma=\lambda w.b(w,s_i)$とする。

ただし、s_iはあるxに対するd(x)の値である。

$$\lambda w.\gamma(b(w,d(x)))$$
$$=\lambda w.b(b(w,d(x)),s_i)$$

したがって、

$$\lambda w.b(b(\ldots b(w,s_n),\ldots,s_2),s_1)$$

となる。これをリスト $[s_n,\ldots,s_2,s_1]$で表現すれば，$\gamma(v)$の計算はつぎの手順で得ることができる。

```
fcsend(v,l) <== if l=[] then v
               else fcsend(b(v,hd(l)),tl(l))
```

最終的にf(x)は以下のとおりとなる。

```
f(x) <== fc1(x,[])
```

```
fc1(x,l) <== if p(x) then fcsend(a(x),l)
                else fc1(e(x),cons(d(x),l))
```

しかし、もしbが右単位元 l_b を有し、結合律を満たすなら、その閉式は

$$b(b(\ldots b(w,s_n)\ldots,s_2),s_1)$$
$$=b(w,b(s_n,\ldots b(s_1,l_b)\ldots)$$

と変形できるから、

```
f(x) <== fc(c,lb)
fc(x,γ) <== if p(x) then b(a(x),γ)
               else fc(c(x),b(d(x),γ))
```

つぎのよく知られた、リストを逆順に並べ換える関数はこの性質を用いて変換することができる例である。

```
Rev(x) <== if x=[] then []
           else append(Rev(tl(x)),[hd(x)])
```

ただし、appendはリストをそのまま連結する関数であり、単位元[]を有し、結合律を満たす。

（３）ｎｏｎｌｉｎｅａｒ－ｓｉｍｐｌｅの場合

つぎの例は最も簡単な場合である。

```
f(x) <== if p(x) then c(x)
         else f(a(f(b(x))))
```

そのContinuation型は

```
fc(x,γ) <== if p(x) then γ(c(x))
               else fc(b(x),λw.γ(f(a(x))))
```

である。Continuationの閉式は、関数の合成を利用すれば次のように表現することができる。すなわち、

$$\lambda w.(f\circ a)^{i}(w)$$
$$=\lambda w.f(a(f(a(\ldots f(a(w))\ldots)))$$

これはカウンタを用いて表現することができる。ただ、γ(a(x))を計算する際にfが出現するので、fcsendの形が少し異なることに注意を要する。

```
f(x) <== fc1(x,0)
fc1(x,i) <== if p(x) then fcsend(c(x),i)
               else fc1(b(x),i+1)
fcsend(v,i) <== if i=0 then v
                  else fc1(a(v),i-1)
```

３．１で述べた MaCathyの９１関数はこの変換の最も単純な例である。

（４）ｎｏｎｌｉｎｅａｒ－ｎｏｔ　ｓｉｍｐｌｅ

再帰関数が入れ子になって出現するかしないかで二通りの場合を扱う。

１）入れ子の場合

```
f(x) <== if p(x) then c(x)
         else f(a(x,f(b(x))))
```

はその例である。このとき、Continuationの閉式は

$$\lambda w.f(a(s_1,f(s_2,\ldots,f(a(s_n,w)),\ldots)))$$

となる。ただし、s_iはあるxの値である。

これはリスト［$s_n,\ldots,s_2,s_1$］で表現すれば、最終的に以下のように変換することができる。

```
f(x) <== fc1(x,[])
fc1(x,l) <== if p(x) then fcsend(c(x),l)
               else fc1(b(x),cons(x,l))
fcsend(v,l) <== if l=[] then v
                  else fc1(a(hd(l),v),tl(l))
```

２）入れ子でない場合

その代表的な例は

```
f(x) <== if p(x) then a(x)
         else b(f(l(x)),f(r(x)))
```

である。Continuationの方法をそのまま適用すると以下のようになる。

```
fc(x,γ) <== if p(X) then γ(a(x))
               else fc(l(x),λw.γ(b(w,f(r(x)))))
```

Continuationの閉式は

$$\lambda w.b(b(\ldots b(w,f(s_n)),\ldots,f(s_2)),f(s_1))$$

ただし、s_iはあるxに対するr(x)の値である。これをリスト［$s_n,\ldots,s_2,s_1$］で表現すれば、結局、つぎのように変換することができる。

```
f(x) <== fc1(x,[])
fc1(x,l) <== if p(x) then fcsend(a(x),l)
               else fc1(l(x),cons(r(x),l))
fcsend(v,l) <== if l=[] then v
                  else b(v,fc1(hd(l),tl(l)))
```

しかし、これはまだ扱い易い形とはいえない。これを関数 l , r あるいは b の性質によって一層扱い易い形に変形するには、Continuationによる方法を拡張する必要がある。その点については節を改めて述べる。

４．Continuationによる方法の拡張

３で述べた手法は、［９］で展開されているところと基本的に異ならないが、二、三の記法の導入によってContinuationの閉式の見通しが得易くなった。その要点は一変数の付加による一般化ということであった。しかし、Continuationをリストで表現せざるを得ない場合、これをさらに効率のよい形に変換したり、一般的変換方法の存在しない nonlinear 型を扱うには、関数の性質および前処理としてContinuation以外の方法も利用する必要がある。このような場合、関数に二個以上の変数を付加する変換の有効なことが少なくない。本節では、これを以下の二つの方法でを扱う。まず、Continuationに基づく方法を二変数以上の一般化に適用するため、これを帰

納法と結合して拡張する。ついで、アキュミュレータに該当する変数の導入によって、あらかじめ、関数を拡張しておき、その後でContinuationを適用する手法を述べる。

4．1　Continuationと帰納法の結合

3節まで、Continuationを

$f(x_1,x_2,\ldots,x_n,\gamma)=\gamma(f(x_1,x_2,\ldots,x_n))$

を満たすものとして扱ってきたが、本節ではこれを拡張し、つぎの関係(※)を満足するものと仮定する。すなわち、結果が$f(x_1,x_2,\ldots,x_n)$も含めた複数個の値の関係によって表現されるものとする。

(※)　$f(x_1,x_2,..,x_n,\gamma)$

$=\gamma(f(x_1,..,x_n),f(\alpha_{11}(x_1),..,\alpha_{1n}(x_n)),$

$f(\beta_{11}(x_1),..,\beta_{1n}(x_n)),...)$

ここで、$\alpha_{11},\ldots,\alpha_{1n},\beta_{11},\ldots,\beta_{1n},\ldots$は既知関数である。

これを、いわゆる、Fibonacci 型の関数に適用し、その変換方法を説明しよう。

$f(x) \Leftarrow$ if $p(x)$ then b_1
else if $p(e(x))$ then b_2
else $h(f(e(x)),f(e^2(x)))$

$e(x)=X$とおくと$f(X,e(X))$であるから、

$fc(x,\gamma)=\gamma(f(x),f(e(x)))$

と仮定すれば、$fc(x,\gamma)$を以下のように変換することができる。

$fc(x,\gamma) \Leftarrow$ if $p(x)$ then $\gamma(b_1)$ else
if $p(e(x))$ then $\gamma(b_2)$ else
$fc(e(x),\lambda uv.\gamma(h(u,v),u))$

このとき、以下のように$fc(x,\gamma)=\gamma(f(x),f(e(x)))$の成立することを証明することができる。

$fc(x,\gamma)=fc(e(x),\lambda uv.\gamma(h(u,v),u))$
$=\lambda uv.\gamma(h(u,v),u)(f(e(x)),f(e^2(x)))$
$=\gamma(h(f(e(x)),f(e^2(x))),f(e(x)))$
$=\gamma(f(x),f(e(x)))$

$\gamma=\lambda st.(h(s,t),s)$とおくと、

$\lambda uv.\gamma(h(u,v),u)$
$=\lambda uv.(\lambda st.(h(s,t),s))(h(u,v),u)$
$=\lambda uv.(h(h(u,v),u),h(u,v))$

となる。すなわち、$(h(u,v),u))$に対し、つぎのλは$(h(h(u,v),u),h(u,v))$である。したがって、最初の$f(x),f(e(x))$の値をそれぞれr,sで表わせば、一般につぎの関係のあることがわかる。

$g_0(r,s)=r$
$g_1(r,s)=s$
$g_2(r,s)=h(s,r)$
$g_3(r,s)=h(h(s,r),s)$

一般に、

$g_{i+1}(r,s)=g_i(s,h(s,r))$

である。これを利用すれば以下の変換結果を得る。

$f(x) \Leftarrow g(x,b_1,b_2)$
$g(x,r,s) \Leftarrow$ if $p(x)$ then r
else if $p((x))$ then s
else $g(e(x),s,h(s,r))$

以上から、Continuation適用の拡張によって、引き続くContinuationの関係を機械的に見付け、これを二個以上の変数を付け加える変換に適用可能なことがわかる。一般に、再帰関数の変換は、引き続く変数の関数適用の関係をいかに機械的に見出すかにかかっている。ここで提案した方法は、［1］の帰納的に関係を見出す方法に比し、より直接的、機械的にその関係を導出する。

4．2　アキュミュレータ変数の利用

nonlinear型の関数をその性質を利用して変換する方法を考察する。

$f(x) \Leftarrow$ if $p(x)$ then $a(x)$
else $b(f(l(x)),f(r(x)))$

そのため、b を左単位元1_bを有し，結合律を満たすと仮定する。このとき、

$G(x,v) = b(v,f(x))$

とおくと

$f(x) = G(x,1_b)$

である。G の定義を少し奇異に感じるかもしれないが、v がプログラム言語のアキュミュレータの役割を果たすことに気がつけばそれほど不自然なものではないことがわかる。

$G(x,v)$
$=b(v,f(x))$
$=b(v,b(f(l(x)),f(r(x))))$　　(f の定義)
$=b(b(v,f(l(x))),f(r(x)))$　　(b の結合律)
$=b(G(l(x),v),f(r(x)))$　　(G の定義)
$=G(r(x),G(l(x),v))$　　(G の定義)

したがって、

$G(x,v) \Leftarrow$ if $p(x)$ then $b(v,a(x))$
else $G(r(x),G(l(x),v))$

となる。これをさらに Continuationによって変換すれば、以下のようになる。

$Gc(x,v,\gamma) \Leftarrow$ if $p(x)$ then $\gamma(b(v,a(x)))$
else $Gc(l(x),v,\lambda w.\gamma(G(r(x),w)))$

γの閉式は、$\lambda w.G(s_n,G(...,G(s_1,w)...))$ である。た

だし、s_i はある xに対する r(x) の値である。これをリスト［s_n,...,s_2,s_1］で表現すれば、３節と同様に変換することができ、以下のとおりとなる。

```
f(x) = Gc1(x,1b,[])
Gc1(x,v,L) <== if p(x) then Gcsend(b(v,a(x)),L)
                 else Gc1(1(x),v,cons(r(x),L))
Gcsend(u,L) <== if L=[] then u
                  else Gc1(hd(L),u,t1(1))
```

ところで、もし 1(x)=e(x),r(x)=e^2(x) であれば、Fibonacci 型になるが、この場合、つぎのような変換を行なうことができる、すなわち、e(1(x))=r(x) であるから、Continuationの閉式は、

λw.G(e^n(1(x)),...,G(e^2(1(x)),G(e(1(x)),w))...)

となる。これは、カウンタを用いて実現することができる。したがって、

```
Gc(x,v,i) <== if p(x) then Gcsend(b(v,a(x),i)
                else Gc(1(x),v,i+1)
Gcsend(u,i) <== if i=0 then u
                  else Gc(e(u),i-1)
```

５．他の方法との比較

ここで述べたContunuationに基づく方法を、二、三の具体例に適用するとともに、既に提案されている他の方法、特に、［１］の方法と比較、検討する。

５．１　Ackermannの関数

この有名な関数を逐次型に変換する方法はよく知られているが［例えば、５］、ここではContinuationを用いてどのように変換されるかを示そう。

```
A(m,n) <== if m=0 then n+1
           else if n=0 then A(m-1,1)
                else A(m-1,A(m,n-1))
```

まず、Continuation-passing 型に変形する。

```
Ac(m,n,γ) <== if m=0 then γ(n+1) else
              if n=0 then Ac(m-1,1,λw.γ(w))
              else Ac(m,n-1,λw.γ(A(m-1,w)))
```

γ=λv.A(s,v)とする。ただし、sはあるmに対する値である。

```
λw.γ(A(m-1,w))
=λw.(λv.A(s,v))(A(m-1,w))
=λw.A(s,A(m-1,w))
```

したがって、その閉じた形は

λw.A(s_1,A(s_2,...,A(s_n,w)...))

となる。これをリスト［s_n,...,s_2,s_1］で表現すると

γ(n+1)=A(s_1,A(s_2,...,(A(s_n,n+1)...))

となる。この計算はつぎの関数によって行なうことができる。

```
Acsend(v,l) <== if l=[] then v else
                Acsend(Ac1(hd(l),v,[]),t1(l))
A(m,n) <== Ac1(m,n,[])
Ac1(m,n,l) <== if m=0 then Acsend(n+1,l)
               else if n=0 then Ac1(m-1,1,l)
                    else Ac1(m,n-1,cons(m-1,l))
```

以上のように、Continuationの方法の機械的適用によって変換されることがわかる。

５．２　特殊なnonlinear型への適用

つぎに［１］で非手続き言語を用いて変換されている関数をContinuationの方法によって変換してみよう。

```
f(n) <== if n=1 then 1
         else if even(n) then f(n/2)
              else h(f((n-1)/2),f((n+1)/2))
```

ここで、hに特別の性質を仮定しない。

これを処理するまえに、これを一般化した次の型の再帰的関数の変換について検討しておくことにする。

```
f(x) <== if p(x) then a(x)
         else if q(x) then b(f(c(x)))
              else d(f(1(x)),f(r(x)))
```

b,dに関してつぎの関係が成立するものと仮定する。

```
b(d(x,y))=d(b(x),b(y))
```

このとき、fのContinuation-passing 型は以下のように特殊な変換が可能となる。

```
f(x,γ) <== if p(x) then γ(a(x)) else
           if q(x) then fc(c(x),λw.γ(b(w)))
           else ?
```

このγの閉式は、３で述べたように、λw.b'(w)となり、これはカウンタを用いて表現することできる。したがって、

```
f(x,i) <== if p(x) then fcsend(a(x),i)
           else if q(x) then fc1(c(x),i+1)
                else ?
```

となる。"？"で示したところはContinuationの仮定により

```
b'(d(f(1(x)),f(r(x))))
=d(b'(f(1(x))),b'(f(r(x))))
=d(fc1(1(x),1),fc1(r(x),i))
```

とならなければならない。最終的に、fは以下のようになる。

```
f(x) <== fc1(x,0)
fc1(x,i) <== if p(x) then fcsend(a(x),i) else
```

```
                if q(x) then fc1(c(x),i+1)
                else d(fc1(l(x),i),fc1(r(x),i))
    fcsend(v,i) <== if i=0 then v
                   else fcsend(b(v),i-1)
```

これは逐次型再帰関数ではない。さらに、変換を行なうには4で述べた方法を利用することになる。

以上の準備のもとで、問題の関数の変換に戻ろう。

bを恒等関数、すなわち、b(x)=xとすれば上で述べた関数に帰着されるから、その変換結果以下のようになる。

```
    f(n) <== fc(n,0)
    fc(n,i) <== if n=1 then 1 else
                if even(n) then fc(n/2,i+1)
                else h(fc((n-1)/2,i),fc((n+1)/2,i))
```

6．おわりに

本論文では、再帰関数を逐次型、あるいは、扱い易い型の再帰関数に変換するため、Continuationを利用する方法を考察した。その方法は、"カン"や経験によるところが少なく、体系的、機械的に適用することができることを示した。とくに、変換の難しいnonlinear型を扱い易くするため、Continuationの適用方法をを拡張した。これによって、一変数の一般化の変形と考えられるContinuationの方法が、二変数以上の関数の一般化にも適用可能となり、従来、"カン"にたよるところの多かった二変数以上の一般化がきわめて自然に導きだされることを示した。また、Continuationによる変換を一層有効なものにするため、適用の前後で利用し得る関数の性質についても種々検討した。

したがって、ここで述べたContinuationによる方法およびその拡張を利用すれば、その他の方法を併用しなくても、相当の範囲の再帰関数に適用することができ、自動変換システムの実現にも利用できると思われる。また、"人手で"行なう変換にも有効である。

ここで述べた方法は、等式言語のプログラム変換システムの実現に用いる予定である。

参考文献

[1] Arsac,J. and Kodratoff,Y. Some techniques for recursion removal from recursive functions TOPLAS 4,2(Apr.1982),295-322

[2] Burstall,R. M. and Darington,J. A transformation system for developing recursive programs J. ACM 24,1(Jan.1977), 44-67

[3] Burstall,R. M. and Darington,J. A system which automatically improves programs Acta Inf. 6(1976), 41-60

[4] Manna,Z. Mathematical theory of computation McGraw-Hill,New York,1974

[5] Manna,Z. and Waldinger,R. Is "sometime" sometimes better than "always"? C.ACM ,21,2(Feb.1978) 159-172

[6] Morris,J. H. and Wegbreit,B. Subgoal Induction C.ACM 20,4(Apr.1977) 209-222

[7] Reynolds,J. C. On the relation between direct and continuation semantics Proc. 2nd Colloq. on Automata, Languages, and Programming, J.Loeckx, Ed., Lecture Notes in Computer Sciences, Vol.14, Springer-Verlag, New York, 1974,pp.141-156

[8] Walker,S. A. and Strong, H. R. Characterization of a flowchartable recursion J. Comput. Syst. Sci. 7(1973),404-447

[9] Wand,M. Continuation-based program transformation strategies J. ACM. 27,1(Jan.1980),164-180

[10] Wegbreit,B. Goal directed program transformation IEEE Trans. Softw. Eng.SE-2,2(June 1976),69-80