2004年度 修士論文

再帰除去システムの実装によるMathematicaの機能拡張

提出日：2005年2月2日

指導：二村 良彦　教授
早稲田大学大学院・理工学研究科・情報ネットワーク専攻
学籍番号：3603U085-5
氏名：竹内 丈志

概 要

再帰プログラムは反復プログラムに比べて読みやすく書きやすいが、計算機で実行する際には再帰呼び出しとスタック操作が必要である。そのため、インライン展開ができない、局所参照性が悪い、などのプログラム最適化上の問題を引き起こす。従って、再帰プログラムを計算量を増加させずにスタックを使用しない反復プログラムに変換する再帰除去法が、1970 年代から研究されてきた。1998 年には、累積関数という概念を用いることでより広範囲に適用できる再帰除去法が、二村良彦教授らによって提案された [1]。本稿では、この累積関数を用いた線形再帰プログラムからの再帰除去法を数式処理ソフト Mathematica 上で実装し、これを Mathematica が備えている既存の処理系と融合することで機能の拡張を目指すものである。
(キーワード)
再帰プログラム，再帰除去，累積関数，Mathematica

Recursive programs are often easier to write and read than iterative ones, but in executing on computers, they require procedure calls and stack operations. This causes problems in program optimizations concerning inline cording and the locality of data references. Therefore, recursion removal methods, transforming a given recursive program into iterative one without using stack and increasing amount of computation time, have been studied since 1970's. In 1998, a recursion removal method based on cumulative functions which can be applicable on a wide scale was proposed by Mr. Futamura[1]. The purpose in this paper is functional expansion of the Mathematica by the implementation of a recursion removal method based on cumulative functions for a linear recursive program which essentially includes only one recursive call in its body.
(Keywords)
recursive program, recursion removal, cumulative functions, Mathematica

目 次

図 目 次

第1章　はじめに

プログラム作成において、プログラムの開発効率よりプログラムの性能効率が重視されることは稀であり、高い技術力を持った熟練プログラマーでないと、プログラムの性能効率を上げることは容易ではない。そこで、最適化されていないプログラムを系統的方法で最適化するために研究されている分野が、プログラム変換がある。
プログラム変換の研究内容や対象は多岐にわたり、再帰除去法・融合法・同素組織化・一般化・部分計算・一般部分計算などがある。本稿で扱う再帰除去法とは、再帰構造を除去することで再帰呼び出しによるメモリ消費や時間コストをなくし、かつ、計算量を増加させることなく反復プログラムに変換するもので、1970 年代中頃から研究されている。また、数値計算の再帰プログラムに関しては反復プログラムから閉式化を行うことで、さらなる計算量の改善が期待できる。
閉式化の方法には、数学的知識を利用した方法 (特性方程式を用いる方法・母関数による方法など) と再帰除去法を応用した方法がある。前者の数学的知識を利用した方法は、数式処理系 Mathematica における再帰方程式の閉式化を行うための組込み関数 RSolve によって既に実装されているが、再帰方程式の形よっては適用不可能であったり、再帰方程式を解くことに重点が置かれているために実用的な解を得られない場合もある。後者の再帰除去法を応用した方法は、累積関数という概念を用いて再帰プログラムを反復プログラムに変換し、さらに閉式化を行う比較的単純なもので、線形再帰プログラム (再帰呼び出しを実質的に 1 箇所でしか行わないプログラム) に対して有効である。
本稿では、累積関数を用いた再帰除去法を線形再帰プログラムについて、数式処理系 Mathematica 上で、先に述べたような既存の関数 RSolve が持つ弱点を克服できるように一部改良を加えて実装し、この関数 RSolve と相互利用することで、より実用的な再帰除去システム (Mathematica 上でのみ動作することを考えると再帰除去関数になる) を実装する。

第2章　再帰除去について

再帰プログラムは反復プログラムに比べて読みやすく書きやすいが、計算機で実行する際には再帰呼び出しとスタック操作が必要である。そのため、インライン展開ができない、局所参照性が悪い、などのプログラム最適化上の問題を引き起こす。従って、再帰プログラムを計算量を増加させずにスタックを使用しない反復プログラムに変換する再帰除去法が、1970 年代から研究されてきた。1998 年には二村教授らによって、それまでより広範囲な線形再帰プログラムに適用できる累積関数を用いた再帰除去法が提案され [1]、さらにこれを拡張し、単一後継関数を持つ一部の非線形再帰プログラムが累積関数持つ場合についても適用可能であることが知られている [2, 5, 7, 8]。数値計算 (用) 再帰プログラムの一部に対しては、累積関数により閉じた式への変換 (閉式化; 即ち漸化式の求解) も可能にした [2, 3]。なお、ここで言う「単一後継関数を持つ再帰プログラム」とは以下の形式のプログラムであり、a, b, c, d は f に対する再帰呼び出しを含まないものとし、それぞれ補助関数・基底関数・制御関数・後継関数と呼ばれる。

$$\begin{aligned} f(x) \quad = \quad & \text{if } p(x) \text{ then } b(x) \text{ else} \\ & a(c(x),\, f(d(x)),\, f(d^2(x)),\, \ldots,\, f(d^n(x))) \end{aligned} \tag{2.1}$$

2.1　累積関数を用いた再帰除去法

累積関数を用いた再帰除去法は、多くの再帰プログラムに対する再帰除去を可能にした [1, 2, 5]。累積関数法は、末尾再帰からの再帰除去、及び、結合性を用いた再帰除去を含む方法である。ここでは、累積関数の定義とそれを用いた再帰プログラムからの再帰除去について紹介する。

2.1.1　累積関数の定義

再帰プログラム (2.1) に対して、下記の性質を持つ関数 $h(v,\, u,\, d(u),\, d^2(u),\, \ldots,\, d^{n-1}(u))$ を a に関する f の累積関数と呼ぶ。
「$\neg p(u)$ ならば任意の式 v に対して、
$a(v,\, f(u),\, f(d(u)),\, f(d^2(u)),\, \ldots,\, f(d^{n-1}(u)))$
$= a(h(v,\, u,\, d(u),\, d^2(u),\, \ldots,\, d^{n-1}(u)),\, f(d(u)),\, f(d^2(u)),\, \ldots,\, f(d^n(u)))$ となる。
ただし、h は f への再帰呼び出し及びその他の自由変数を含んではならず、$p(d^i(x))$ を満たす最小の自然数 i を N とするとき、$N + n > i > N$ なる任意の i に対して $p(d^i(x))$ は成立するものとする。」

2.1.2 再帰プログラムからの再帰除去

定理

累積関数を持つ再帰プログラム (2.1) は、以下のような反復プログラム $floop(x)$ に変換できる。

$floop(x) =$
if $p(x)$ **then** **return** $b(x)$ **else**
begin
  $v := c(x)$; $w[1] := d(x)$;
  for $i := 2$ **to** n **do** $w[i] := d(w[i-1])$;
  while not $p(w[1])$ **do**
  begin
    $v := h(v, w[1], \ldots, w[n])$;
    for $i := 1$ **to** $n-1$ **do** $w[i] := w[i+1]$;
    $w[n] := d(w[n])$;
  end;
  return $a(v, b(w[1]), \ldots, b(w[n]))$;
end

証明

与えられた x に対して、$f(x)$ の計算が停止する場合をまず考える。この時 $p(x)$ ならば、$f(x) = floop(x) = b(x)$ は明らか。$\neg p(x)$ ならば、ある $N = \min\{i | 1 \leq i \wedge p(d^i(x))\}$ が存在する。よって、

$$
\begin{aligned}
& f(x) \\
= \quad & a(c(x), f(d(x)), \ldots, f(d^n(x))) \\
= \quad & a(h(c(x), d(x), \ldots, d^n(x)), f(d^2(x)), \ldots, f(d^{n+1}(x))) \\
= \quad & a(h(h(c(x), d^2(x), \ldots, d^n(x)), d^2(x), \ldots, d^{n+1}(x)), f(d^3(x)), \ldots, f(d^{n+2}(x))) \\
= \quad & \ldots \\
= \quad & a(h(\ldots h(h(c(x), d(x), \ldots, d^n(x)), d^2(x), \ldots, d^{n+1}(x)), \ldots), b(d^N(x)), \ldots, b(d^{N+n-1}(x)))
\end{aligned}
$$

これを最左最内規則で計算すると、以下の計算列が得られる。

$v := c(x)$; $u_1 := d(x)$; $u_2 := d(u_1)$; $\ldots$ $u_n := d(u_{n-1})$;
$v := h(v, u_1, \ldots, u_n)$; $u_1 := u_2$; $u_2 := u_3$; $\ldots$ $u_{n-1} := u_n$; $u_n := d(u_n)$;
$\vdots$

$v := h(v, u_1, \ldots, u_n); \quad u_1 := u_2; \quad u_2 := u_3; \quad \ldots \quad u_{n-1} := u_n; \quad u_n := d(u_n);$
return $a(v, b(d^N(x)), \ldots, b(d^{N+n-1}(x)))$

この計算列は上記の反復プログラムと同じ計算を行う。$f(x)$ が停止しない場合、$floop(x)$ も明らかに停止しない。

2.2 再帰プログラムの閉式化

数値計算プログラムにおいては、再帰除去のみならず、閉じた式にすることが望まれる。閉じた式を求めることで、constant time で計算できるプログラムへ変換可能になる。

2.2.1 分類定理の拡張

理論計算量を評価するに当たって行われている方法は、「漸化式を作る」「漸化式を解く」という作業によるものであるが、この「漸化式を解く」という作業 [9] については、古くから分類定理という方法が提案されている。この分類定理を拡張し、より汎用性のある漸化式 (プログラム) として以下のような式 (2.2) について、累積関数を用いた再帰除去を応用することで一定の変換規則が得られる [3]。ここでは、その定理について紹介する。

$$\begin{aligned} f(x) \quad = \quad & \text{if } p(x) \text{ then } b(x) \\ & \text{else } c_1(x) + c_2(x) * f(d(x)) \end{aligned} \tag{2.2}$$

定理

式 (2.2) は、式 (2.3) へ変換可能である。

$$\begin{aligned} f(x) \quad = \quad & c_1(x) + c_2(x) * \sum_{i=1}^{N} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} \\ & + b(d(u_N)) * c_2(x) * \prod_{k=1}^{N} c_2(u_k) \end{aligned} \tag{2.3}$$

ただし、$u_0 = x$, $u_m = d(u_{m-1})$
また、$p(d(u_n))$ により N の式を n の式にする必要がある。

証明

$f(x)$ は累積関数を用いた再帰除去により、以下のように変換可能である。

$f(x) =$
if $p(x)$ **then** $b(x)$ **else**

begin

$v_1 := c_1(x);\quad v_2 := c_2(x);\quad u := x;$

while not $p(d(u))$ **do**

begin

$u := d(u);\quad v_1 := v_1 + v_2 * c_1(u);\quad v_2 := v_2 * c_2(u);$

end;

return $v_1 + v_2 * b(d(u))$;

end

そして、v_1^k, v_2^k, u_k をそれぞれ k 回目のループの v_1, v_2, u とすると、

$$\begin{array}{ccc} v_1^0 = c_1(x) & v_2^0 = c_2(x) & u_0 = x \\ v_1^1 = v_1^0 + v_2^0 * c_1(u_m) & v_2^1 = v_2^0 * c_2(u_1) & u_1 = d(u_0) \\ \vdots & \vdots & \vdots \\ v_1^m = v_1^{m-1} + v_2^{m-1} * c_1(u_m) & v_2^m = v_2^{m-1} * c_2(u_m) & u_m = d(u_{m-1}) \end{array}$$

また、ループが停止した時の u を u_N とすると、$p(d(u_N))$ が成り立つ。ここで、u_N が解けたとすると、

$$\begin{aligned} v_1^N &= v_1^0 + \sum_{i=1}^{N} v_2^{i-1} * c_1(u_i) \\ &= c_1(n) + c_2(n) * \sum_{i=1}^{N} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} \\ v_2^N &= c_2(n) * \prod_{k=1}^{N} c_2(u_k) \end{aligned}$$

となる。したがって、

$$\begin{aligned} f(x) &= v_1^N + v_2^N * b(d(u_N)) \\ &= c_1(x) + c_2(x) * \sum_{i=1}^{N} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} \\ &\quad + b(d(u_N)) * c_2(x) * \prod_{k=1}^{N} c_2(u_k) \end{aligned}$$

ただし、$p(d(u_N))$ により N の式を n の式にする必要がある。

第3章　Mathematicaが抱える問題点とその解決法

Mathematica には、再帰方程式を解くための既存の処理系 RSolve が存在する。この処理系 RSolve の内部実装に関しては、以下のようになっている。

- RSolve は定数係数を持つ線形方程式系を行列のベキ乗を使って解く。
- 多項式係数を持つ、解が超幾何的に与えられる線形方程式は van Hoeij 法で解く。
- 有理関数の係数を持つ、解が有理関数として与えられる線形方程式の系は Abramov-Bronstein 消去法で解く。
- 非線形方程式は変数の変換，Göktas シンメトリー消去法，あるいは Germundsson 三角ベキ法が使われる。
- Mathematica のアルゴリズムは，数学の文献でこれまでに論じられたほとんどの一般的な方程式と q 階差分方程式をカバーしている。
- 差分代数方程式では，核ベキ零分解による特異部分の分離に基づく方法が使われる。

RSolve による再帰方程式の解法は強力で適用範囲も広いが、後継関数の形には一部制限があり、また、「再帰法方程式を数学的に解いて解を求める」ことに重点が置かれているため、求められた解を利用するに当たって実用的でない場合も存在する。ここでは、既存の処理系 RSolve を利用することでより実用的な再帰除去システム (図 3.1 参照) の実装を行うために、RSolve が抱えるいくつかの問題についてその解決法と共に述べる。なお、実装する再帰除去システムの動作の仕方については、各々の再帰方程式について RSolve の動作の仕方に準ずるものとする。

3.1　基底条件式の自動生成

3.1.1　問題点 1

図 3.2 は、RSolve にフィボナッチ数列を表す再帰方程式

$$
\begin{aligned}
&f(x) = \\
&\textbf{if } x \leq 1 \quad \textbf{then } 1 \\
&\textbf{else } f(x-1) + f(x-2)
\end{aligned}
$$

を与えて実行したものである。図 3.2 中の式 $\mathbf{fib}[0] == 1$, $\mathbf{fib}[1] == 1$ が基底条件 (式) であるが、不等号を用いて式を表わすことができないので、与える再帰方程式の形に応じてその数を必要な分だけ記述してやらなければならない。

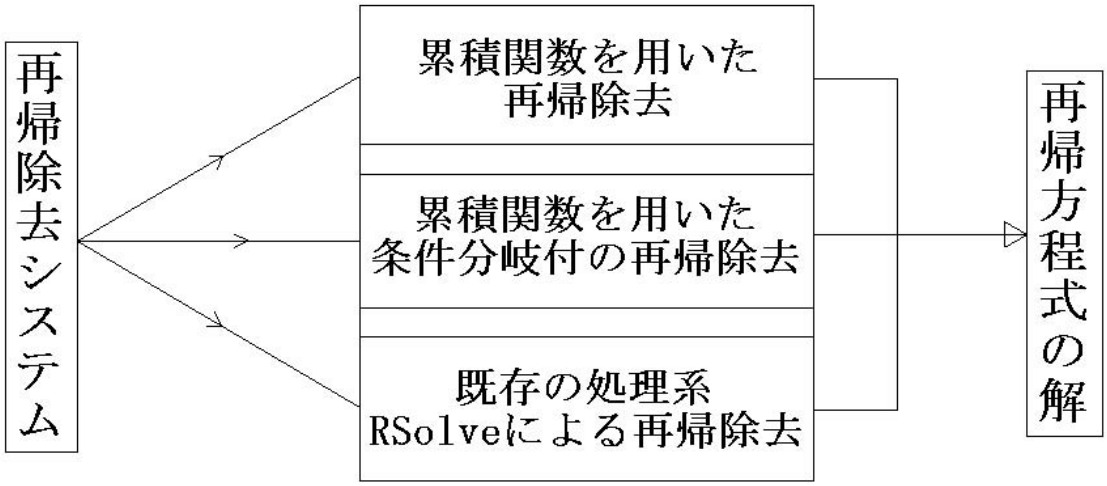

図 3.1: 実装する再帰除去システム

```
RSolve[{f[x] == f[x - 1] + f[x - 2], f[0] == 1, f[1] == 1}, {f}, {x}]
```

$$\left\{\left\{f \to \text{Function}\left[\{x\},\ \frac{1}{10}\left(5\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^{x}-\sqrt{5}\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^{x}+5\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^{x}+\sqrt{5}\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^{x}\right)\right]\right\}\right\}$$

図 3.2: RSolve のフィボナッチ数列を表す再帰方程式に対する実行例

3.1.2 解決法

再帰方程式における不等号で表された基底条件より、基底条件式の自動生成を行う。これは式 (2.1) における、基底条件 p・基底関数 b・後継関数 d から生成できる。具体的には、Mathematica が扱える後継関数の形によって以下のように分類し、それぞれの場合について Mathematica の動作の仕方に基づいて基底条件式を与える。

$d(x) = x + k,\ k \in Z$ の場合

$p(x)$ を $x \leq h$ または $x \geq h$ とする。$(h \in R)$

- 線形再帰方程式の場合
  必要となる基底条件式の基底関数 b が引数とする x の値は、以下の $|k|$ 個についてである。この x の値それぞれについて、基底関数 b を適用して基底条件式を求める。

$$\begin{aligned} x_1 &= h \\ x_2 &= h + 1 * \frac{k}{|k|} \\ x_3 &= h + 2 * \frac{k}{|k|} \\ &\vdots \\ x_{|k|} &= h + (|k| - 1) * \frac{k}{|k|} \end{aligned}$$

- 非線形再帰方程式の場合
  複数ある後継関数 d において、$|k|$ が最大のものについて線形再帰方程式と同じ処理を行い、基底条件式を求める。

$d(x) = x * k,\ k \in Q$ の場合

$p(x)$ を $x \leq h$ または $x \geq h$ とする。$(h \in R)$

- 線形再帰方程式の場合
  必要となる基底条件式の基底関数 b が引数とする x の値は、$x_1 = h$ の 1 つである。したがって、基底関数 b を適用した式 $b(x) = b(h)$ が基底条件式となる。

- 非線形再帰方程式の場合
  複数ある後継関数 d を $d(x) = x * k,\ d^2(x) = x * k^2,\ \ldots,\ d^n(x) = x * k^n$ とすると、必要となる基底条件式の基底関数 b が引数とする x の値は、以下の n 個についてである。この x の値それぞれについて、基底関数 b を適用して基底条件式を求める。

$$\begin{aligned} x_1 &= h \\ x_2 &= h * k \\ x_3 &= h * k^2 \end{aligned}$$

$$\vdots$$
$$x_n \quad = \quad h * k^n$$

3.2 後継関数の適用範囲拡張

```
RSolve[{f[x] == √x * f[√x] + x, f[2] == 1}, {f}, {x}]

RSolve::piarg : f[x] == x + √x f[√x]の位置1の引数はすべてx + Integerまたはq^
   Integer * xという形式でなくてはなりません. これらの形式は混在してはなりません. 詳細

RSolve[{f[x] == x + √x f[√x], f[2] == 1}, {f}, {x}]
```

図 3.3: RSolve の後継関数が $d(x) = x^k,\ k \in Q$ である再帰方程式に対する実行例

3.2.1 問題点 2

図 3.3 は、RSolve に再帰方程式

$$f(x) =$$
$$\textbf{if } x \leq 2 \quad \textbf{then } 1$$
$$\textbf{else } \sqrt{x} * f(\sqrt{x}) + x$$

を与えて実行したものである。再帰方程式の後継関数の形が $d(x) = x^k,\ k \in Q$ のとき、RSolve ではこの再帰方程式を解くことができない。

3.2.2 解決法

式 (2.2) から式 (2.3) への変換規則を用いることで解決できる。

3.3 条件分岐付解の自動生成

3.3.1 問題点 3

図 3.4 は、RSolve に再帰方程式

$$f(x) =$$
$$\textbf{if } x \leq 2 \quad \textbf{then } x$$
$$\textbf{else } f(x-3) + x$$

```
R = RSolve[{f[x] == f[x - 3] + x, f[2] == 2, f[1] == 1, f[0] == 0}, {f}, {x}][[1, 1, 2]]
Function[{x},
 (-6 + 6 (-1)^(1/3) + 12 (-1)^(2/3) - 60 i Sqrt[3] - 60 (-1)^(5/6) Sqrt[3] + 45 x - 45 (-1)^(1/3) x - 90 (-1)^(2/3) x +
    9 x^2 - 9 (-1)^(1/3) x^2 - 18 (-1)^(2/3) x^2 + 6 Cos[(2 Pi x)/3] - 6 (-1)^(1/3) Cos[(2 Pi x)/3] -
    12 (-1)^(2/3) Cos[(2 Pi x)/3] + 18 i Sqrt[3] Cos[(2 Pi x)/3] + 18 (-1)^(5/6) Sqrt[3] Cos[(2 Pi x)/3] -
    18 (-1)^(1/6 + (2 x)/3) Sqrt[3] Cos[(2 Pi x)/3] + 18 i (-1)^(1/3 + (2 x)/3) Sqrt[3] Cos[(2 Pi x)/3] +
    12 (-1)^(5/6 + (2 x)/3) Sqrt[3] Cos[(2 Pi x)/3] + 12 i (-1)^(2 x/3) Sqrt[3] Cos[(2 Pi x)/3] +
    30 i Sqrt[3] (-(-1)^(1/3))^x Cos[(2 Pi x)/3] + 18 (-1)^(1/6) Sqrt[3] (-(-1)^(1/3))^x Cos[(2 Pi x)/3] +
    12 (-1)^(5/6) Sqrt[3] (-(-1)^(1/3))^x Cos[(2 Pi x)/3] - 6 (-1)^(1/6 + (2 x)/3) Sqrt[3] x Cos[(2 Pi x)/3] +
    6 i (-1)^(1/3 + (2 x)/3) Sqrt[3] x Cos[(2 Pi x)/3] + 6 (-1)^(5/6 + (2 x)/3) Sqrt[3] x Cos[(2 Pi x)/3] +
    6 i (-1)^(2 x/3) Sqrt[3] x Cos[(2 Pi x)/3] + 12 i Sqrt[3] (-(-1)^(1/3))^x x Cos[(2 Pi x)/3] +
    6 (-1)^(1/6) Sqrt[3] (-(-1)^(1/3))^x x Cos[(2 Pi x)/3] + 6 (-1)^(5/6) Sqrt[3] (-(-1)^(1/3))^x x Cos[(2 Pi x)/3] +
    18 i Sin[(2 Pi x)/3] + 18 (-1)^(5/6) Sin[(2 Pi x)/3] - 36 i (-1)^(2/3 + (2 x)/3) Sin[(2 Pi x)/3] +
    54 i (-1)^(2 x/3) Sin[(2 Pi x)/3] + 6 Sqrt[3] Sin[(2 Pi x)/3] - 6 (-1)^(1/3) Sqrt[3] Sin[(2 Pi x)/3] -
    12 (-1)^(2/3) Sqrt[3] Sin[(2 Pi x)/3] - 36 (-1)^(1/6) (-(-1)^(1/3))^x Sin[(2 Pi x)/3] +
    54 (-1)^(5/6) (-(-1)^(1/3))^x Sin[(2 Pi x)/3] - 18 i (-1)^(2/3 + (2 x)/3) x Sin[(2 Pi x)/3] +
    18 i (-1)^(2 x/3) x Sin[(2 Pi x)/3] - 18 (-1)^(1/6) (-(-1)^(1/3))^x x Sin[(2 Pi x)/3] +
    18 (-1)^(5/6) (-(-1)^(1/3))^x x Sin[(2 Pi x)/3]) /
  (54 (-i + (-1)^(1/6))^2 (i + (-1)^(1/6))^2 (-1 + (-1)^(1/3))^2)]
```

図 3.4: RSolve の後継関数が $d(x) = x + k,\ k \in Z$ である再帰方程式に対する実行例

を与えて実行したものである。再帰方程式の後継関数の形を $d(x) = x + k,\ k \in Z \wedge |k| > 1$ で与えたとき、RSolve はこの後継関数を $d^{|k|}(x) = x + k,\ k \in Z \wedge |k| > 1$ と解釈する。さらに RSolve は、「再帰方程式を数学的に解き、解を 1 つの閉式として表す」ことに重点が置かれている。そのため、求められる解を理解することは非常に困難となり、場合によっては RSolve で求められる解の方が、元の再帰方程式よりもパフォーマンスが落ちてしまうこともある。

3.3.2 解決法

累積関数を用いた再帰除去により求められた式 (2.3) を利用する。これを応用し、式 (2.3) における N について、それぞれ $\mathrm{mod}(x,k) = 0,\ \mathrm{mod}(x,k) = 1,\ \ldots,\ \mathrm{mod}(x,k) = |k| - 1$ の各場合ごとに $N_0,\ N_1,\ \ldots,\ N_{|k|-1}$ を求め、それを適用した条件分岐付の解を求める。(式 (3.1) 参照)

$$
\begin{aligned}
f(x) \quad = \quad & \\
& \text{if } \mathrm{mod}(x,|k|) = 0 \\
& \text{then } c_1(x) + c_2(x) * \sum_{i=1}^{N_0} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} + b(d(u_{N_0})) * c_2(x) * \prod_{k=1}^{N_0} c_2(u_k) \\
& \text{else if } \mathrm{mod}(x,|k|) = 1 \\
& \text{then } c_1(x) + c_2(x) * \sum_{i=1}^{N_1} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} + b(d(u_{N_1})) * c_2(x) * \prod_{k=1}^{N_1} c_2(u_k) \\
& \vdots \\
& \text{else if } \mathrm{mod}(x,|k|) = |k| - 2 \\
& \text{then } c_1(x) + c_2(x) * \sum_{i=1}^{N_{|k|-2}} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} + b(d(u_{N_{|k|-2}})) * c_2(x) * \prod_{k=1}^{N_{|k|-2}} c_2(u_k) \\
& \text{else } \mathrm{mod}(x,|k|) = |k| - 1 \\
& \text{then } c_1(x) + c_2(x) * \sum_{i=1}^{N_{|k|-1}} \left\{ c_1(u_i) * \prod_{k=1}^{i-1} c_2(u_k) \right\} + b(d(u_{N_{|k|-1}})) * c_2(x) * \prod_{k=1}^{N_{|k|-1}} c_2(u_k)
\end{aligned}
\tag{3.1}
$$

第4章　再帰除去システムの効果

実装した再帰除去システムの動作確認と、これによって求められた解と、再帰方程式及び RSolve による解とで、実行時間を比較した。実行環境は「IBM-ThinkPad(X-31), IntelPentiumMprocessor(1400MHz), 1.00GBRAM」上で行った。

4.1　出力結果

4.1.1　問題点 1 について

```
f[x_] :=
 If[x ≤ 1, 1,
  f[x - 1] + f[x - 2]]

SolveRExp[{"f[x]"}]
```

$$\left\{\text{Function}\left[\{x\}, \frac{1}{10}\left(-\left(\frac{1}{2}\left(1-\sqrt{5}\right)\right)^{x}\left(-5+\sqrt{5}\right)+\left(\frac{1}{2}\left(1+\sqrt{5}\right)\right)^{x}\left(5+\sqrt{5}\right)\right)\right]\right\}$$

図 4.1: 実装した再帰除去システムのフィボナッチ数列に対する実行例

4.1.2　問題点 2 について

```
f[x_] := If[x ≤ 2, 1, √x * f[√x] + x]

F = SolveRExp[{"f[x]"}][[1]]

Solve::ifun : 逆関数がSolve
   で使われているため，求められない解がある可能性があります．解の詳細情報にはReduceをお使いください． 詳細

Solve::ifun : 逆関数がSolve
   で使われているため，求められない解がある可能性があります．解の詳細情報にはReduceをお使いください． 詳細

Solve::ifun : 逆関数がSolve
   で使われているため，求められない解がある可能性があります．解の詳細情報にはReduceをお使いください． 詳細

General::stop : 計算中，Solve::ifunのこれ以上の出力は表示されません． 詳細

Function[{x}, 1/Log[4] (x (Log[2] - 2 Log[Log[2]] + 2 Log[Log[x]]))]
```

図 4.2: 実装した再帰除去システムの後継関数が $d(x) = x^k,\ k \in Q$ である再帰方程式に対する実行例

4.1.3 問題点 3 について

```
f[x_] :=
 If[x ≤ 2, 1,
  f[x - 3] + x]

F = SolveRExp[{"f[x]"}][[1]]

Function[{x}, If[Mod[x, 3] == 0, 1/6 (6 + 3 x + x^2),
   If[Mod[x, 3] == 1, 1/6 (2 + 3 x + x^2), 1/6 (-4 + 3 x + x^2)]]]
```

図 4.3: 実装した再帰除去システムの後継関数が $d(x) = x + k,\ k \in Z$ である再帰方程式に対する実行例

4.2 実行時間の比較

4.2.1 問題点 2 について

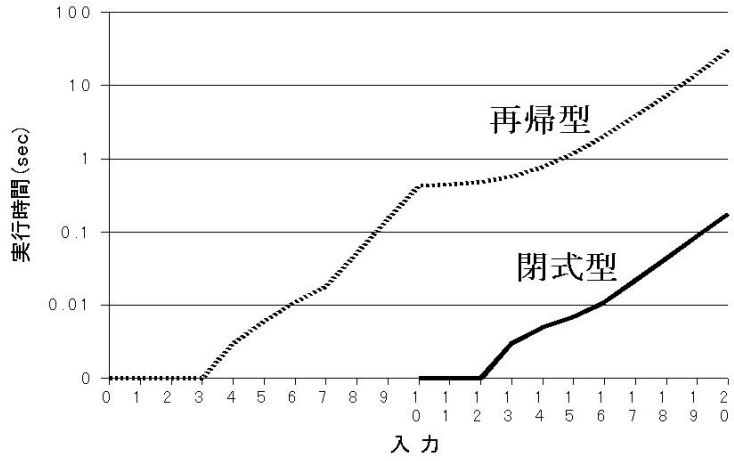

図 4.4: 問題点 2 に対する再帰方程式と閉式の実行時間の比較

4.2.2 問題点 3 について

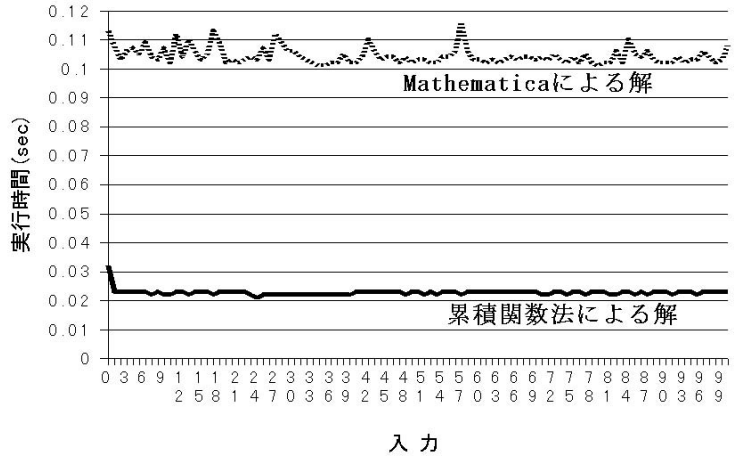

図 4.5: 問題点 3 に対する Mathematica による解と累積関数法による解の実行時間の比較

第5章　おわりに

今回は、累積関数を用いた再帰除去法を用いることで、Mathematica にある再帰方程式の解法用の処理系 (RSolve) の機能拡張を行った。特に本稿で示した問題点 3 については、既存の処理系 (RSolve) が後継関数 d を $d(x) = x + 1 * (k/|k|),\ d^2(x) = x + 2 * (k/|k|),\ \ldots,\ d^{|k|}(x) = x + k, \quad k \in Z$ と解釈し、非線形再帰方程式として計算及び 1 つに閉式化することから発生する問題で、解が数学的で非実用的なものであるといえる。今回はこの点を修正することで、既存の処理系による解よりも、プログラム的で実用的な解を得ることが可能となった。ただし、今回の拡張は主に式 (2.2) の形の線形再帰方程式にのみ適用できるものであり、この形以外の線形再帰方程式、さらには非線形再帰方程式についても適用できるよう、より汎用性を持つ再帰除去システムに改良していく必要がある。

関連図書

[1] 二村良彦，大谷啓記:線形再帰プログラムからの再帰除去とその実際的効果, コンピュータソフトウェア,Vol.15,No.3(1998)

[2] 市川裕輔, 松谷将寛, 小西善二郎, 二村良彦:単一後継関数を持つ再帰プログラムからの再帰除去, 情報学会論文誌,Vol.0,No.0(2001)

[3] 坂本巨樹, 二村良彦:分割統治法に基づくアルゴリズムの計算量自動評価の試み, 日本ソフトウェア科学会第 15 回大会論文集 (1998)

[4] 松谷将寛:線形再帰方程式の累積関数を用いた求解法, 早稲田大学大学院理工学研究科情報科学専攻 2000 年度修士論文 (2001)

[5] 市川裕輔, 小西善二郎, 二村良彦:単一後継関数を持つ再帰プログラムからの再帰除去及び閉式化,FIT(情報科学技術フォーラム)2002

[6] 市川裕輔, 小西善二郎, 二村良彦:再帰除去におけるタプリング法の限界と累積関数法, 日本ソフトウェア科学会第 19 回大会論文集 (2002)

[7] Yusuke ICHIKAWA, Zenjiro KONISHI, Yoshihiko FUTAMURA:Recursion Removal from-Recursive Program with Only One Descent Function,IEICE TRANS.INF.& SYST.,Vol.E86-D,No.3(MARCH,2003)

[8] Yusuke Ichikawa:Cumulative Method(Recursion Removal from Recursive Programs with One Descent Function),2003.Master Thesis,Department of Information and Computer Science.Graduate School of Science and Engineering.Waseda University.

[9] Sedgewick R.,Flajolet P.:An Introduction to the Analysis of Algorithms, Addison Wesley(Reading MA),1996

付 録A　再帰除去システムのソースコード (Mathematica)

```
SolveRExp[FDList_] := If[Length[FDList] == 1 &&
      (Length[Cases[SrcDnExp[SrcFncNameSub[First[FDList]], FDList],
          First[SrcArgName[First[FDList]]]^_]] > 0 ||
       Length[Cases[SrcDnExp[SrcFncNameSub[First[FDList]], FDList],
           First[SrcArgName[First[FDList]]] + _]] == 1 &&
        Length[CheckBCase[First[FDList], First[SrcArgName[First[FDList]]],
           SrcDnExp[SrcFncNameSub[First[FDList]], FDList]]] > 1),
     {RtnTnSExp[FDList, SrcFncNameSub[First[FDList]],
       First[SrcArgName[First[FDList]]]]},
     SmplfyRExp[Extract[RSolve[Join[MkMainExp[FDList, SrcFncName[FDList]],
         MkBCaseExp[FDList, FDList]], AddDummyString[SrcFncName[FDList]],
        SrcArgName[First[FDList]]], {1}]]]

SrcDnExp[FncName_, FDList_] := If[FDList == {}, {},
     Union[SrcDnExpSub[First[FDList], EditRCallPosition[
        SrcRCallPosition[FncName, First[FDList], {}]]],
      SrcDnExp[FncName, Delete[FDList, 1]]]]

SrcDnExpSub[FncDef_, PList_] := If[PList == {}, {},
     Union[RtnDnExp[FncDef, First[PList], Length[SrcArgName[FncDef]]],
      SrcDnExpSub[FncDef, Delete[PList, 1]]]]

RtnDnExp[FncDef_, PList_, ArgLng_] := If[ArgLng == 0, {},
     Append[RtnDnExp[FncDef, PList, ArgLng - 1],
      Extract[ToExpression[FncDef], Append[PList, ArgLng]]]]

SrcArgName[FncDef_] := SrcArgNameSub2[FncDef, SrcArgNameSub1[
      StringPosition[FncDef, {"[", ",", "]"}]]]

SrcArgNameSub2[FncDef_, PositionList_] := If[PositionList == {}, {},
     Append[SrcArgNameSub2[FncDef, Delete[PositionList, 1]],
      ToExpression[StringTake[FncDef, First[PositionList]]]]]
```

```
SrcArgNameSub1[PositionList_] := If[Length[PositionList] == 1, {},
     Append[SrcArgNameSub1[Delete[PositionList, 1]],
      {PositionList[[1,2]] + 1, PositionList[[2,1]] - 1}]]

EditRCallPosition[PList_] := If[PList == {}, {}, If[First[PList] == 0,
      Prepend[EditRCallPosition[Delete[PList, 1]], {}],
      Prepend[Delete[EditRCallPosition[Delete[PList, 1]], 1],
       Prepend[First[EditRCallPosition[Delete[PList, 1]]], First[PList]]]]]

SrcRCallPosition[FncName_, FncDef_, PList_] :=
    Switch[ToString[Extract[ToExpression[FncDef], Append[PList, 0]]], "If",
     SrcRCallPosition[FncName, FncDef, Append[PList, 3]], "Plus",
     SrcRCallPositionSub[FncName, FncDef, PList,
      Length[Extract[ToExpression[FncDef], PList]]], "Times",
     If[ToString[Extract[ToExpression[FncDef], Append[PList, 1]]] == "-1",
      SrcRCallPositionSub[FncName, FncDef, PList, 2],
      SrcRCallPositionSub[FncName, FncDef, PList,
       Length[Extract[ToExpression[FncDef], PList]]]], "Power",
     SrcRCallPositionSub[FncName, FncDef, PList,
      Length[Extract[ToExpression[FncDef], PList]]], ToString[FncName],
     Append[PList, 0], _, NULL]

SrcRCallPositionSub[FncName_, FncDef_, PList_, ArgLng_] :=
    If[ArgLng == 0, NULL, If[ArgLng == 1, SrcRCallPosition[FncName, FncDef,
       Append[PList, ArgLng]], If[ToString[SrcRCallPosition[FncName, FncDef,
          Append[PList, ArgLng]]] == "NULL",
       If[ToString[SrcRCallPositionSub[FncName, FncDef, PList,
           ArgLng - 1]] == "NULL", NULL, SrcRCallPositionSub[FncName, FncDef,
         PList, ArgLng - 1]], If[ToString[SrcRCallPositionSub[FncName,
           FncDef, PList, ArgLng - 1]] == "NULL", SrcRCallPosition[FncName,
         FncDef, Append[PList, ArgLng]], Join[SrcRCallPositionSub[FncName,
          FncDef, PList, ArgLng - 1], SrcRCallPosition[FncName, FncDef,
          Append[PList, ArgLng]]]]]]]

SrcFncNameSub[FncDef_] := ToExpression[StringTake[FncDef,
      StringPosition[FncDef, "["][[1,1]] - 1]]

CheckBCase[FncDef_, ArgName_, dnList_] :=
    If[Length[Cases[dnList, ArgName + _]] > 0,
     If[First[Cases[dnList, ArgName + _] - ArgName] < 0,
      MkBCaseSub1[SrcPnExp[FncDef], SrcFncNameSub[FncDef],
       SrcArgName[FncDef], First[Cases[dnList, ArgName + _] - ArgName]],
      MkBCaseSub1[SrcPnExp[FncDef], SrcFncNameSub[FncDef],
```

```
       SrcArgName[FncDef], Last[Cases[dnList, ArgName + _] - ArgName]]],
     If[Length[Cases[dnList, ArgName*_]] > 0,
      If[Length[Cases[dnList, ArgName*_]] == 1, MkBCaseSub2[SrcPnExp[FncDef],
        SrcFncNameSub[FncDef], SrcArgName[FncDef],
        First[Cases[dnList, ArgName*_]/ArgName]],
       MkBCaseSub3[SrcPnExp[FncDef], SrcFncNameSub[FncDef],
        SrcArgName[FncDef], Cases[dnList, ArgName*_]/ArgName]],
      If[Length[Cases[dnList, ArgName^_]] > 0, "Constructing....",
       "ERROR!!"]]]

MkBCaseSub1[Pn_, FncName_, ArgName_, dNum_] := Switch[ToString[Head[Pn]],
     "LessEqual", If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      Table[{i, FncName[i]}, {i, Extract[Pn, {2}] + dNum + 1,
        Extract[Pn, {2}]}], Table[{i, FncName[i]}, {i, Extract[Pn, {1}],
        Extract[Pn, {1}] + dNum - 1}]], "GreaterEqual",
     If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      Table[{i, FncName[i]}, {i, Extract[Pn, {2}], Extract[Pn, {2}] + dNum -
         1}], Table[{i, FncName[i]}, {i, Extract[Pn, {1}] + dNum + 1,
        Extract[Pn, {1}]}]], "Equal", If[ToString[Extract[Pn, {1}]] ==
       ToString[First[ArgName]], {{Extract[Pn, {2}],
        FncName[Extract[Pn, {2}]]}}, {{Extract[Pn, {1}],
        FncName[Extract[Pn, {1}]]}}], _, "ERROR(MkBCaseSub1)"]

SrcPnExp[FncDef_] := If[ToExpression[FncDef][[0]] == If,
     ToExpression[FncDef][[1]], "Error!!:The Function is wrog form."]

MkBCaseSub2[Pn_, FncName_, ArgName_, dNum_] := Switch[ToString[Head[Pn]],
     "LessEqual", If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      {{Extract[Pn, {2}], FncName[Extract[Pn, {2}]]}},
      {{Extract[Pn, {1}], FncName[Extract[Pn, {1}]]}}], "GreaterEqual",
     If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      {{Extract[Pn, {2}], FncName[Extract[Pn, {2}]]}},
      {{Extract[Pn, {1}], FncName[Extract[Pn, {1}]]}}], "Equal",
     If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      {{Extract[Pn, {2}], FncName[Extract[Pn, {2}]]}},
      {{Extract[Pn, {1}], FncName[Extract[Pn, {1}]]}}], _,
     "ERROR(MkBCaseSub2)"]

MkBCaseSub3[Pn_, FncName_, ArgName_, dnList_] := Switch[ToString[Head[Pn]],
     "LessEqual", If[ToString[Extract[Pn, {1}]] == ToString[First[ArgName]],
      MkBCaseList[FncName, Extract[Pn, {2}], Last[dnList],
       Log[Last[dnList], First[dnList]]], MkBCaseList[FncName,
       Extract[Pn, {1}], First[dnList], Log[First[dnList], Last[dnList]]]],
```

```
     "GreaterEqual", If[ToString[Extract[Pn, {1}]] ==
       ToString[First[ArgName]], MkBCaseList[FncName, Extract[Pn, {2}],
       First[dnList], Log[First[dnList], Last[dnList]]],
      MkBCaseList[FncName, Extract[Pn, {1}], Last[dnList],
       Log[Last[dnList], First[dnList]]]], "Equal", "ERROR(MkBCaseSub3)", _,
     "ERROR(MkBCaseSub3)"]

MkBCaseList[FncName_, BCPoint_, dNum_, Count_] :=
    If[Count == 0, {}, Append[MkBCaseList[FncName, BCPoint*dNum, dNum,
       Count - 1], {BCPoint, FncName[BCPoint]}]]

RtnTnSExp[FDList_, FncName_, ArgName_] :=
    If[Length[Cases[SrcDnExp[FncName, FDList], ArgName^_]] > 0,
     Function[{ArgName}, Evaluate[Simplify[PowerExpand[MkTnSExpSub[FncName,
          ArgName, Extract[SrcDnExp[FncName, FDList], {1, 2}],
          RtnPnNum[First[FDList], ArgName]]]]], Function[{ArgName},
      Evaluate[ToExpression[RtnTnSExpSub[FncName, ArgName,
         Extract[SrcDnExp[FncName, FDList], {1, 1}],
         CheckBCase[First[FDList], ArgName, SrcDnExp[FncName, FDList]]]]]]]

MkTnSExpSub[FncName_, ArgName_, dNum_, pNum_] :=
    SrcC1Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName] +
     MkTnSPart1Sub[FncName, ArgName, dNum, pNum] + MkTnSPart2Sub[FncName,
      ArgName, dNum, pNum]

SrcC1Exp[FncDef_, FncName_, ArgName_] := Extract[FncName[ArgName], {3}] -
     FncName[Extract[SrcDnExp[FncName, FncDef], {1}]]*
      Coefficient[Extract[FncName[ArgName], {3}],
       FncName[Extract[SrcDnExp[FncName, FncDef], {1}]]]

MkFncDefString[FncName_, ArgExp_] := StringJoin[ToString[FncName], "[",
     ToString[StandardForm[ArgExp]], "]"]

MkTnSPart1Sub[FncName_, ArgName_, dNum_, pNum_] :=
    SrcC2Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName]*
     Sum[SrcC1Exp[{MkFncDefString[FncName, MkUExpSub[ArgName, dNum, i]]},
        FncName, MkUExpSub[ArgName, dNum, i]]*
       Product[SrcC2Exp[{MkFncDefString[FncName, MkUExpSub[ArgName, dNum,
            k]]}, FncName, MkUExpSub[ArgName, dNum, k]], {k, 1, i - 1}],
      {i, 1, MkLoopEndNSub[ArgName, dNum, pNum, N]}]

SrcC2Exp[FncDef_, FncName_, ArgName_] :=
    Coefficient[Extract[FncName[ArgName], {3}],
```

```
     FncName[Extract[SrcDnExp[FncName, FncDef], {1}]]]

MkUExpSub[ArgName_, dNum_, loopT_] := ArgName^dNum^loopT

MkLoopEndNSub[ArgName_, dNum_, pNum_, loopT_] :=
    Extract[Solve[MkUExpSub[ArgName, dNum, loopT] == pNum, loopT],
      {1, 1, 2}] - 1

MkTnSPart2Sub[FncName_, ArgName_, dNum_, pNum_] :=
    FncName[ArgName^dNum^(MkLoopEndNSub[ArgName, dNum, pNum, N] + 1)]*
     SrcC2Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName]*
     Product[SrcC2Exp[{MkFncDefString[FncName, MkUExpSub[ArgName, dNum, k]]},
       FncName, MkUExpSub[ArgName, dNum, k]],
      {k, 1, MkLoopEndNSub[ArgName, dNum, pNum, N]}]

RtnPnNum[FncDef_, ArgName_] :=
    If[ToString[Extract[First[ToExpression[FncDef]], {1}]] ==
      ToString[ArgName], Extract[First[ToExpression[FncDef]], {2}],
     Extract[First[ToExpression[FncDef]], {1}]]

RtnTnSExpSub[FncName_, ArgName_, dNum_, BCList_] :=
    If[Length[BCList] == 0, StringRepeat["]", Abs[dNum] - 1],
     If[Length[BCList] == 1, StringJoin[ToString[StandardForm[
         Simplify[PowerExpand[Evaluate[MkTnSExp[FncName, ArgName, dNum,
             Extract[BCList, {1, 1}]]]]]]], RtnTnSExpSub[FncName, ArgName,
       dNum, Drop[BCList, 1]]], StringJoin["If[Mod[", ToString[ArgName],
       ",", ToString[Abs[dNum]], "]==", ToString[StandardForm[
         Mod[Extract[BCList, {1, 1}], Abs[dNum]]]], ",",
       ToString[StandardForm[Simplify[PowerExpand[Evaluate[MkTnSExp[FncName,
             ArgName, dNum, Extract[BCList, {1, 1}]]]]]]], ",",
       RtnTnSExpSub[FncName, ArgName, dNum, Drop[BCList, 1]]]]]

StringRepeat[Str_, Lng_] := If[Lng == 1, Str, StringJoin[Str,
      StringRepeat[Str, Lng - 1]]]

MkTnSExp[FncName_, ArgName_, dNum_, pNum_] :=
    SrcC1Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName] +
     MkTnSPart1[FncName, ArgName, dNum, pNum] + MkTnSPart2[FncName, ArgName,
      dNum, pNum]

MkTnSPart1[FncName_, ArgName_, dNum_, pNum_] :=
    SrcC2Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName]*
     Sum[SrcC1Exp[{MkFncDefString[FncName, MkUExp[ArgName, dNum, i]]},
```

```
        FncName, MkUExp[ArgName, dNum, i]]*
       Product[SrcC2Exp[{MkFncDefString[FncName, MkUExp[ArgName, dNum, k]]},
         FncName, MkUExp[ArgName, dNum, k]], {k, 1, i - 1}],
      {i, 1, MkLoopEndN[ArgName, dNum, pNum, N]}]

MkUExp[ArgName_, dNum_, loopT_] := ArgName + dNum*loopT

MkLoopEndN[ArgName_, dNum_, pNum_, loopT_] :=
    Extract[Solve[MkUExp[ArgName, dNum, loopT] == pNum, loopT], {1, 1, 2}] - 1

MkTnSPart2[FncName_, ArgName_, dNum_, pNum_] :=
    FncName[ArgName + dNum*(MkLoopEndN[ArgName, dNum, pNum, N] + 1)]*
     SrcC2Exp[{MkFncDefString[FncName, ArgName]}, FncName, ArgName]*
     Product[SrcC2Exp[{MkFncDefString[FncName, MkUExp[ArgName, dNum, k]]},
       FncName, MkUExp[ArgName, dNum, k]], {k, 1, MkLoopEndN[ArgName, dNum,
        pNum, N]}]

SmplfyRExp[AnsList_] := If[AnsList == {}, {},
     Append[SmplfyRExp[Drop[AnsList, {1}]], SmplfyRExpSub[
       Extract[AnsList, {1, 2}]]]]

SmplfyRExpSub[RFnc_] := Function[Evaluate[Extract[RFnc, {1}]],
     Evaluate[Simplify[Extract[RFnc, {2}]]]]

MkMainExp[FDList_, FNList_] := If[FDList == {}, {},
     Prepend[MkMainExp[Delete[FDList, 1], FNList],
      Extract[MkMainExpSub[First[FDList], FNList], {1}] ==
       Extract[MkMainExpSub[First[FDList], FNList], {2}]]]

MkMainExpSub[FncDef_, FNList_] :=
    {ToExpression[StringReplace[FncDef, ToString[SrcFncNameSub[FncDef]] ->
        ToString[AddDummyStringSub[SrcFncNameSub[FncDef]]]]],
     Extract[ToExpression[StringReplace[ToString[StandardForm[
          SrcMainExp[FncDef]]], MkTFRule[FNList]]], {1}]}

AddDummyStringSub[FncName_] := ToExpression[StringJoin[ToString[FncName],
      "Dummy"]]

SrcMainExp[FncDef_] := If[ToString[ToExpression[FncDef][[0]]] == "If",
     Extract[ToExpression[FncDef], 3, Hold],
     "Error!!:The Function is wrong form."]

MkTFRule[FNList_] := If[FNList == {}, {},
```

```
     Prepend[MkTFRule[Delete[FNList, 1]], ToString[First[FNList]] ->
       ToString[First[AddDummyString[FNList]]]]]

AddDummyString[FNList_] := If[FNList == {}, {},
     Prepend[AddDummyString[Delete[FNList, 1]], AddDummyStringSub[
       First[FNList]]]]

SrcFncName[FDList_] := If[FDList == {}, {},
     Prepend[SrcFncName[Delete[FDList, 1]], SrcFncNameSub[First[FDList]]]]

MkBCaseExp[FncDef_, FDList_] := If[FncDef == {}, {},
     Join[MkBCase[First[FncDef], CheckBCase[First[FncDef],
        First[SrcArgName[First[FncDef]]], SrcDnExp[SrcFncNameSub[
          First[FncDef]], FDList]]], MkBCaseExp[Delete[FncDef, 1], FDList]]]

MkBCase[FncDef_, BCList_] := If[BCList == {}, {},
     Append[MkBCase[FncDef, Delete[BCList, 1]],
      AddDummyStringSub[SrcFncNameSub[FncDef]][Extract[First[BCList],
         {1}]] == Extract[First[BCList], {2}]]]
```