再帰プログラム部分計算のための停止性判定法

宋 立彤[†] 二村良彦[††] Robert Glück[††]

本稿ではまず，部分計算に関する新しい停止性判定法である再帰条件法を提案する．再帰条件法は，既存の停止性判定法である実引数法と同じく，同相埋め込み（homeomorphic embedding）関係を利用した整列擬順序（well-quasi ordering）に基づく判定法である．実引数法が，実引数の文字列的な順序に関する逆順を調べるのに対して，再帰条件法は，再帰呼出しが起こる場合の条件（再帰条件）に関する指標の文字列的な逆順を調べる．したがって，両者とも，任意の部分計算を必ず止めることのできる停止性判定条件であるが，止まるタイミングは同じとは限らない．どちらが先に止まるかは，実行される部分計算によって異なる．しかし，再帰条件法は意味論的な情報をより多く持つ再帰条件を用いるため，多くの場合に，実引数法よりある意味で都合良く止まることができる．本稿ではこの2つの停止条件の利害得失について詳細に議論する．さらに，本稿では再帰条件法と実引数法の長所を取り入れた混合法を提案する．混合法は，この両者の停止条件をともに満たす場合にのみ停止するので，いかなる場合でも両者よりも早く止まる可能性がない判定法である．

A Termination Condition for Partial Evaluation of Recursive Program

Litong Song,[†] Yoshihiko Futamura[††] and Robert Glück[††]

In this paper, we present a new termination condition called recursion condition method (RCM) to ensure the termination of specialization. Just like existing termination conditions which we call as actual parameter method or APM, RCM is also a termination condition based on well-quasi ordering using homeomorphic embedding. Compared with using the actual parameters of functions or procedures in APM, the recursion conditions invoking recursive calls are used in RCM. If APM and RCM are used respectively in the specialization of a program, the residual programs may be different for the different termination timing of the two termination conditions. Which termination condition makes specialization terminate earlier or later differs according to source program. However, because RCM uses the recursion conditions containing semantic information of a source program to some extent, it can often produce more efficient residual programs than APM does. Furthermore, we can combine RCM with APM into a new termination condition called combined method or CM. Only if the termination conditions of RCM and APM are satisfied simultaneously, the termination condition of CM is satisfied, so CM will not allow specialization process to terminate earlier than both RCM and APM.

1. はじめに

部分計算は，原始プログラムとその計算に必要な実引数の一部を受け取り，既知の値に関してプログラムを特化したより効率の良いプログラム（剰余プログラム）を生成するプログラム変換である．部分計算は，本番計算にとっては一種の前処理であるので，実用的には必ず停止させる必要がある．さらに理想的には，我々が欲しい停止性判定法は，最適の剰余プログラムを最短時間で生成可能にするものである．

しかし，再帰プログラムの停止性判定が決定不能であるのと同じく，再帰プログラムの部分計算の停止性判定問題も決定不能であるので，上記のような理想的判定条件で決定可能なものは存在しえない．そこで現実的には，できるだけ能率の良い剰余プログラムを，できるだけ短い時間で生成することのできる，決定可能な停止性判定法を発見する必要がある．

我々の調査の限りでは，既存の部分計算器は，停止性判定法として限界終了法（bounded termination）または実引数法（actual parameter）を利用している．

限界終了法は，部分計算の実行時間あるいは剰余プログラムのサイズ等が所定の量を超えたら部分計算を終了させる方法である．その方法に基づいた部分計算は，原始プログラムの意味論的特徴とは無関係に早め

† 早稲田大学大学院理工学研究科
Graduate School of Science and Engineering, Waseda University
†† 早稲田大学理工学部情報学科
Department of Information and Computer Science, Waseda University

に止まるか，あるいは不必要に部分計算をやりすぎることにより，剰余プログラム最適化の機会を逃す場合が多い（Cmix[1]，Similix[2]，Tempo[4]）．一方実引数法は，原始（再帰）プログラムの実引数に基づいて整列順序（WFO）あるいは整列擬順序（WQO）を定義し，この順序に基づくある意味での逆順が再帰呼出しの実引数として出現するまで部分計算を続けるものである．それは文献 3)，9)，14)，15)，18)〜22) 等の部分計算器あるいは部分演繹器で使われた．また，部分計算あるいは部分演繹等の動的停止性判定法として実引数法を用いる際には，WQO が WFO より有力であることが文献 14) において証明されている．文献 3)，9)，14)，15)，19)〜22) では，そこで採用された対象言語がそれぞれパターン・マッチ機能を持つ関数型言語 Refal と論理型言語 Prolog であるため，原始プログラムの意味論的特徴の一部が引数に反映される．そのため，上述の部分計算器に関しては，実引数法は限界終了法よりも比較的良いタイミングで停止可能である．しかし，パターン・マッチ機能を持たない対象言語に対しては，実引数法においても限界終了法と同様に停止が早すぎたり，あるいは遅すぎるという場合が頻繁に発生する．

本稿では，同じ原始プログラムに対する 2 つの剰余プログラムを比べて，時間と空間に関する能率の悪いプログラムは最適化エラーが大きいという．逆に，能率の良い剰余プログラムは最適化エラーが小さいと呼ぶ．さらに最適化エラーが大きい原因が，停止が早すぎたことに起因する場合には，停止性判定法にはやり残しが多いと呼ぶ．逆に，停止が遅すぎたことに起因する場合には停止性判定法にはやりすぎが多いと呼ぶ．たとえば，下記のように定義された再帰関数 p(x,y) について考える：

$$\begin{aligned} \mathrm{p}(\mathrm{x},\mathrm{y}) \stackrel{\mathrm{def}}{=} & \underline{\mathrm{if}}\ \mathrm{x} \le -100\ \underline{\mathrm{then}}\ \mathrm{x} \\ & \underline{\mathrm{else}}\ \mathrm{if}\ \mathrm{x} > 100\ \underline{\mathrm{then}}\ \mathrm{y} \\ & \quad \underline{\mathrm{else}}\ \mathrm{p}(\mathrm{x}+1,\mathrm{y}-1) \end{aligned}$$

このプログラムを最適化したもの p(x,y) は次のとおりである：

$$\begin{aligned} \mathrm{o\_p}(\mathrm{x},\mathrm{y}) \stackrel{\mathrm{def}}{=} & \underline{\mathrm{if}}\ \mathrm{x} \le -100\ \underline{\mathrm{then}}\ \mathrm{x} \\ & \underline{\mathrm{else}}\ \underline{\mathrm{if}}\ \mathrm{x} > 100\ \underline{\mathrm{then}}\ \mathrm{y} \\ & \quad \underline{\mathrm{else}}\ \mathrm{y}+\mathrm{x}-101 \end{aligned}$$

したがって，引数 x が未知の場合には，部分計算により p が展開される回数が増えるほど剰余プログラムが大きくなり，やりすぎが多くなる．逆に，x が既知の場合には，p(x,y) の部分計算が続けば続くほどやり残しが少なくなる．この例の部分計算に関しては，実引数法は本稿の後の部分で提案される再帰条件法に比べてやり残しまたはやりすぎを非常に起こしやすい．ちなみに，現在提案されている部分計算法の中で，上記の p(x,y) を o-p(x,y) に自動変換できるものは GPC[6)〜8)] のみであるが，本稿では GPC のような定理証明を利用した部分計算器は比較の対象から除外する．本稿では，決定可能な停止性判定手続きに関する議論のみを行うことが除外の理由である．

本稿で提案する同相埋め込み関係を利用した WQO に基づく再帰条件法は，実引数の代わりに，再帰プログラムにおける再帰呼出しを起こす条件を利用する．そのため，再帰条件法は，実引数法よりも再帰呼出しに敏感となり，最適化エラーの小さい剰余プログラムの生成を可能とする．再帰条件法はパターン・マッチ機能を持つ対象言語に対しても適用可能であるが，本稿ではパターン・マッチ機能を持たない関数型言語のみを対象として議論する．さらに，我々は再帰条件法と実引数法を組み合わせた混合法を提案する．混合法は，両者のどちらよりも早く止まることがないので，やりすぎのリスクを冒しながらも，やり残しを大幅に減らしたい場合には有効である．

2. 整列擬順序と埋め込み関係

整列擬順序は，項書換規則[5),16)]，部分計算[9),19),21),22)] および部分演繹[3),14),15)] 等の停止性に関する議論でよく使われる．本章では整列擬順序に関する既知の定義と定理を紹介する．

定義 1（擬順序）任意の集合 S に対して，$\mathrm{S}\times\mathrm{S}$ 上の関係 $\preccurlyeq$ が反射的かつ推移的であるとき，$\preccurlyeq$ を S 上の**擬順序**と呼ぶ．

擬順序が反対称性を持つならば部分順序になることに注意されたい．

定義 2（許容列）任意の集合 S に対して $\preccurlyeq$ を S 上の擬順序とする．S の要素の列 $\mathrm{s}_1,\mathrm{s}_2,\ldots$ において，$1 \le \mathrm{i} < \mathrm{j}$ かつ $\mathrm{s_i} \preccurlyeq \mathrm{s_j}$（以後，この関係を**逆順**と呼ぶ）なる i，j が存在しないときに限り，この列を $\preccurlyeq$ に関する**許容列**（admissible sequence）と呼ぶ．

定義 3（整列擬順序 $\preccurlyeq$）関係 $\preccurlyeq$ を S 上の擬順序とする．この関係に関する S 上のすべての許容列が有限長であるとき，関係 $\preccurlyeq$ を S 上の**整列擬順序**と呼ぶ．

以下では，整列擬順序を WQO と略記する．本稿における以下の議論で利用する整列擬順序である同相埋め込み関係[10),11)] を次のように定義する．

定義 4（同相埋め込み関係 $\preccurlyeq_{\mathrm{emb}}$）$\preccurlyeq$ を S 上の擬順序とする．また，$\mathrm{S_{emb}} = \{\mathrm{a_1a_2}\ldots\mathrm{a_n} \mid \mathrm{a_i} \in \mathrm{S}, 1 \le \mathrm{n}\}$

とする．このとき下記の関係 $\preccurlyeq_{emb}$ を $\preccurlyeq$ による $S_{emb} \times S_{emb}$ 上の**同相埋め込み関係**と呼ぶ．

$$a_1 a_2 \ldots a_n \preccurlyeq_{emb} b_1 b_2 \ldots b_m \quad \text{iff } \forall j_{(1 \le j \le n)} \cdot (a_j \preccurlyeq b_{ij})$$

ただし，$1 \le i_1 < i_2 \ldots < i_n \le m$.

定理 1 $\preccurlyeq$ が S 上の WQO ならば，同相埋め込み関係 $\preccurlyeq_{emb}$ も S_{emb} 上の WQO である．

定義 5（直積擬順序 $\precsim$）$\preccurlyeq_1, \ldots, \preccurlyeq_n$ がそれぞれ $S_1, \ldots, S_n$ 上の擬順序とする．また，$S^n = S_1 \times \ldots \times S_n$ とする．このとき下記の関係 $\precsim$ を $\preccurlyeq_1, \ldots, \preccurlyeq_n$ による S^n 上の**直積擬順序**と呼ぶ．

$$(a_1, a_2, \ldots, a_n) \precsim (b_1, b_2, \ldots, b_n) \quad \text{iff } \forall i_{(1 \le i \le n)} \cdot (a_i \preccurlyeq_i b_i)$$

定理 2 $\preccurlyeq_1, \ldots, \preccurlyeq_n$ がそれぞれ $S_1, \ldots, S_n$ 上の WQO ならば，直積擬順序 $\precsim$ は S^n 上の WQO である．

定理 1 と定理 2 の証明は文献 10)，17) を参照されたい．

定義 6（最大 N 次元直積擬順序 $\lesssim$）$\preccurlyeq_1, \ldots, \preccurlyeq_n$ がそれぞれ $S_1, \ldots, S_n$ 上の擬順序とする．このとき下記の $\lesssim$ を $\preccurlyeq_1, \ldots, \preccurlyeq_n$ による $\bigcup_{n=1}^{N} S^n$ 上の**最大 N 次元直積擬順序**と呼ぶ．

$$(a_1, \ldots, a_m) \lesssim (b_1, \ldots, b_n) \quad \text{iff } \forall i_{(1 \le i \le m = n \le N)} \cdot (a_i \preccurlyeq_i b_i)$$

系 1 $\preccurlyeq_1, \ldots, \preccurlyeq_n$ がそれぞれ $S_1, \ldots, S_n$ 上の WQO ならば，直積擬順序 $\lesssim$ は $\bigcup_{n=1}^{N} S^n$ 上の WQO である．

系 1 の証明は定理 2 に基づいて簡単にできるため，ここでは省略する．従来の再帰条件法に用いられる代表的な埋め込み関係は $\trianglelefteq$ [13),14),19)] である．

定義 7（埋め込み関係 $\trianglelefteq$）式上の埋め込み関係 $\trianglelefteq$ は次のように帰納的に定義される：

(1) $X \trianglelefteq Y$ for all variables X, Y

(2) $s \trianglelefteq f(t_1, \ldots, t_n)$ if $s \trianglelefteq t_i$ for some i

(3) $f(s_1, \ldots, s_n) \trianglelefteq f(t_1, \ldots, t_n)$ if $\forall i_{(i \le i \le n)} \cdot (s_i \trianglelefteq t_i)$

3. 再帰プログラムの定義

本稿で扱う再帰プログラム（関数）を McCarthy の条件式を用いて以下に定義する．

定義 8（再帰プログラム）再帰関数 f(X) は，f 以外の関数に対する再帰呼出しを含まず，かつ次のような形をした関数である．

$$f(X) \stackrel{def}{=} [\neg b(X) \to a(X); b(X) \to E;]$$

ただし，

(1) 式の計算規則は最左最内規則，すなわち call-by-value semantics を仮定する．また，副作用はないものとする．

(2) X は変数ベクトル $< x_1, x_2, \ldots, x_m >$ でもよい．

(3) a は f への再帰呼出を含まない式である．

(4) E は次の 3 種類の式とする．

(a) $f(X, f(d_1(X)), \ldots, f(d_n(X)))$．ただし，$d_i$ $(1 \le i \le n)$ は条件式でない任意の式である．

(b) $g(X, f(d_1(X)), \ldots, f(d_n(X)))$．ただし，g は f 以外の任意の非再帰関数である．

(c) $[\neg b(X) \to E_1; b(X) \to E_2;]$．ただし，$E_1$，$E_2$ の定義は E と同じである．

(5) プログラムで扱うデータ・タイプは Int∪Float∪Char∪Bool∪List∪String のみとする．

(6) 再帰条件式は f への再帰呼出しを含まない積和標準形論理式である．また，本稿で扱う 2 項関係式は $e_1 \oplus e_2$ あるいは $\neg(e_1 \oplus e_2)$ の形式のもののみとする．ただし，e_1 と e_2 は同じ上述のデータ・タイプを持つ任意の式である．そして，$\oplus \in \{=, <, >, \neq, \le, \ge, \subset, \supset, \subseteq, \supseteq\}$ とする．

任意の n 項関係式 $\oplus(e_1, \ldots, e_n)$ に対して，$\oplus(e_1, \ldots, e_n)$ を 2 項関係式に変換することができることに注意されたい．たとえば，$\oplus(e_1, \ldots, e_n)$ を $\oplus(e_1, \ldots, e_n) = \text{true}$ のように変換すればよい．したがって，本稿で用いる関係式は 2 項関係式のみとしても一般性を失わない．

4. 条件指標

我々の提案する再帰条件法は，部分計算の際に，計算を止めるか否かを再帰条件に対応する条件指標と呼ばれる量を見て決定する．

定義 9（条件指標関数，条件指標）b を任意の論理式，そして S をある種の WQO が定義されている整列擬順序集合とする．このとき，b を S の要素に対応させる写像 M_S を S 上の**条件指標関数**と呼ぶ．また $M_S(b)$ を S に関する b の**条件指標**と呼ぶ．

再帰条件法では，条件指標の決め方が本質的に重要である．すなわち，やり残しとやりすぎの両者がともに少なくなるように条件指標関数を定める必要がある．以下では再帰条件（論理式）の複雑さに応じて，対応する条件指標を決める 1 つの条件指標関数を与える．

4.1 2 項関係式に対する条件指標関数

再帰条件として 2 項関係式 $e_1 \oplus e_2$ を持つ計算においては，その計算途上で e_1 と e_2 の差異が確実に減少し，最後にそれらが一致するならば，その計算は必

ず停止する．なぜならば，(1) 2 項関係 $\oplus$ が非反射的（たとえば $<$）な場合 e_1 と e_2 が一致すれば明らかに再帰条件は満たされない．(2) 2 項関係 $\oplus$ が反射的（たとえば $\leq$）な場合，e_1 と e_2 が一致すればその時点では再帰条件が満たされるので停止しない．しかし e_1 と e_2 の差異が確実に減少するという条件により次の時点では停止条件を満たさなければならない（e_1 と e_2 が一致したときに差異が最小になることに注意）．そうでないと，次の時点では $e_1 = e_2$ になるか $e_1 \neq e_2$ となるかのどちらかであり，差異の単調減少性に矛盾する．この停止性のための十分条件は素朴ではあるが，実験的に調べてみると非常に有効であった．そのため，我々は e_1 と e_2 の値の “差” を同相埋め込み関係と条件指標を用いて下記のとおり一般化した．ここではまず，2 項関係式に対する条件指標関数 H を定義する．

定義 10（相違要素数）任意のリスト L に対して，L が未知変数あるいは [] の場合に，$\mathrm{len}(L) \stackrel{\mathrm{def}}{=} 0$，L が [H|T] の場合に，$\mathrm{len}(L) \stackrel{\mathrm{def}}{=} 1 + \mathrm{len}(T)$ とする．任意の 2 つのリスト L_1 と L_2 に対して，(値の定されていない変数を無視して）既知要素の最長共通部分列の長さを k としたとき，$\mathrm{len}(L_1) + \mathrm{len}(L_2) \cdot 2k$ の値を L_1 と L_2 の**相違要素数**と呼ぶ．

たとえば，$L_1 = [a, b, c, b, d|T]$，$L_2 = [b, d, c, a|T]$ とすれば（T は未知），L_1 と L_2 の最長共通部分列 (b, c) の長さは 2，相違要素数は $5 + 4 - 2 \times 2 = 5$ になる．

集合 α を {‘+’, ‘−’, ‘.’, ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘U’} とすると，2 項関係式のための条件指標関数は次のように定義される．

定義 11（2 項関係式のための条件指標関数）H：2 項関係式 $\to \alpha^*$，任意の 2 項関係式 r（$e_1 \oplus e_2$ あるいは $\neg(e_1 \oplus e_2)$）に対して，

$$\mathrm{H}(r) \stackrel{\mathrm{def}}{=} \begin{cases} \text{“U”} & \text{if r の値} = \text{false} \\ & \text{or } e_1, e_2 \in \mathrm{Int} \cup \\ & \mathrm{Float} \cup \mathrm{Char} \cup \\ & \mathrm{Bool} \wedge \text{r の値} \\ & \neq \text{false} \\ t(e_1 - e_2) & \text{if } e_1, e_2 \in \mathrm{Int} \cup \\ & \mathrm{Float} \\ t(ord(e_1) - ord(e_2)) & \text{if } e_1, e_2 \in \mathrm{Char} \\ & \cup \mathrm{Bool} \\ t(p(lst(e_1), lst(e_2))) & \text{if } e_1, e_2 \in \\ & \mathrm{String} \\ t(p(e_1, e_2)) & \text{if } e_1, e_2 \in \mathrm{List} \end{cases}$$

ただし，$lst(s)$ は文字列 s のリスト表現，$t(v)$ は整数あるいは浮動点小数 v の符号付き十進表現，$p(L_1, L_2)$ はリスト L_1 と L_2 の相違要素数とする．

たとえば，$\mathrm{H}(10 > 1) =$ “+9”，$\mathrm{H}([a, a, b] \neq []) =$ “+3”，$\mathrm{H}(x > 1) =$ “U”．

定義 12（同相埋め込み関係 $\preccurlyeq_{\mathrm{emb}}$）$\alpha^* \times \alpha^*$ 上の同相埋め込み関係 $\preccurlyeq_{\mathrm{emb}}$ は α 上の擬順序 $=$ により次のように定義される：

$$a_1 a_2 \ldots a_n \preccurlyeq_{\mathrm{emb}} b_1 b_2 \ldots b_m \quad \text{iff } \forall j_{(1 \leq j \leq n)} \exists i_{(1 \leq i_j \leq m)} \cdot (a_j = b_{ij})$$

たとえば，“−1” $\preccurlyeq_{\mathrm{emb}}$ “−10” かつ “+5” $\preccurlyeq_{\mathrm{emb}}$ “+351” であるが，“+11” と “+12” の間には関係 $\preccurlyeq_{\mathrm{emb}}$ は成立しない．

α が有限集合なので，$=$ は α 上の WQO となる．定理 1 により，$\preccurlyeq_{\mathrm{emb}}$ は α^* 上の WQO となる．

4.2 一般論理式に対する条件指標関数

本節では，再帰条件が和標準形論理式および積和標準形論理式の場合の条件指標関数を定義する．前者の条件指標としては，論理和の要素（2 項関係式）に対する条件指標の積（カルテシアンプロダクト）を用いる．また，後者の条件指標としては論理積の要素（和標準形論理式）に対する条件指標の積を用いる．これらを用いる主な理由は，逆順が現れるタイミングをできるだけ遅らせることである．

(1) 和標準形論理式

定義 13（和標準形論理式に対する条件指標関数）H_o：和標準形論理式 $\to \bigcup_{n=1}^{N} (\alpha^*)^n$．任意の和標準形論理式 $o(r_1 \vee \ldots \vee r_n, 1 \leq n \leq N)$ に対して，

$$H_o(o) \stackrel{\mathrm{def}}{=} (H(r_1), \ldots, H(r_n))$$

定義 14（擬順序 $\precsim_o$）$\bigcup_{n=1}^{N} (\alpha^*)^n$ 上の擬順序 $\precsim_o$ を α^* 上の擬順序 $\preccurlyeq_{\mathrm{emb}}$ により次のように定義する：

$$(a_1, \ldots, a_m) \precsim_o (b_1, \ldots, b_n) \quad \text{iff } \forall_i (1 \leq i \leq m = n \leq N) \cdot (a_i \preccurlyeq_{\mathrm{emb}} b_i)$$

系 1 により $\precsim_o$ は $\bigcup_{n=1}^{N} (\alpha^*)^n$ 上の WQO となる．

たとえば $H_o(0 > 10 \vee 2 < 9) =$ (“U”, “−7”)，$H_o(x_1 > x_2 \vee 10 > 0) =$ (“U”, “+10”)．この 2 つの直積値の間には関係 $\precsim_o$ は成立しない．

(2) 積和標準形論理式

定義 15（積和標準形論理式に対する条件指標関数）H_b：積和標準形論理式 $\to \bigcup_{m=1}^{M} (\bigcup_{n=1}^{N} (\alpha^*)^n)^m$．任意の積和標準形論理式 $b(o_1 \wedge \ldots \wedge o_m, 1 \leq m \leq M)$ に対して，

$$H_b(b) \stackrel{\mathrm{def}}{=} (H_o(o_1), \ldots, H_o(o_m))$$

定義 16（擬順序 $\precsim_b$）$\bigcup_{m=1}^{M} (\bigcup_{n=1}^{N} (\alpha^*)^n)^m$ 上の擬順序 $\precsim_b$ を $\bigcup_{n=1}^{N} (\alpha^*)^n$ 上の擬順序 $\precsim_o$ を用いて次の

ように定義する：

$$(a_1,\ldots,a_p) \precsim b(b_1,\ldots,b_q) \quad \text{iff } \forall i_{(1\le i\le p=q\le N)} \cdot (a_i \precsim_o b_i)$$

系 1 により，$\precsim_b$ は $\bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の WQO となる．

たとえば，$H_b(10>0\wedge 2<9)=((\text{“}+10\text{”}),(\text{“}-7\text{”}))$, $H_b(x_1>x_2\wedge 2<9)=((\text{“U”}),(\text{“}-7\text{”}))$. この 2 つの直積値の間に関係 $\precsim_b$ は成立しない．

再帰条件法は，各再帰呼出しに対する再帰条件に対応する条件指標に注目する．したがって，再帰呼出しおよび停止への条件分岐がプログラムの中に複数個ある場合には，各分岐を 1 つの積和標準形論理式 b（たとえば $B_1\wedge\ldots\wedge B_n$）で表し，一連の通し番号を付ける（たとえば b に番号 NO を付けて b_{NO} にする）．この付け方は，一意的に付けられる限り，順番は問題ではない．そして，その番号と分岐条件を対にして新しい条件指標を下記の関数 H_1 により定義する．

定義 17（条件指標関数 H_1）H_1：論理式 $\to \text{Int}\times\bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$，任意の前述のような通し番号を付けられた分岐条件式 b_{NO} に対して，

$$H_1(b_{NO}) \stackrel{\text{def}}{=} (NO, H_b(b_{NO}))$$

定義 18（擬順序 $\precsim_1$）$\text{Int}\times\bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の関係 $\precsim_1$ は Int 上の擬順序 $=$ と $\bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の擬順序 $\precsim_b$ により次のように定義される：

$$(a_1,a_2)\precsim_1(b_1,b_2) \text{ iff } a_1=b_1\wedge a_2\precsim_b b_2$$

任意の 2 つの異なる条件式 b_i と b_j に対して，$i\neq j$ ならば，$\precsim_1$ の意味で $H_1(b_i)$ と $H_1(b_j)$ とは無関係である．一方，分岐条件式に付けられる番号の集合が当然有限集合なので，擬順序 $=$ は WQO である．系 1 により，擬順序 $\precsim_1$ は $\text{Int}\times\bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の WQO である．

5. 関数呼出し流れ図と部分計算の停止性

5.1 関数呼出し流れ図とその構成

再帰条件法は，再帰呼出しが発生するための再帰条件に着目する．再帰関数の部分計算において，任意の計算経路で出現した条件列の条件指標列が $\precsim_1$ に関する許容列であるか否かにより，その経路の計算が続くか止まるかが決められる．その過程を明確に表現するために，本稿は CFC と呼ばれるラベル付き関数呼出し流れ図を導入する．これは GPC 木[7] と非常によく似た木構造であるが，再帰条件のみならず，再帰呼出しの順序を考慮に入れている点が GPC 木と異なる．議論をしやすくするために，まず 2 つの概念を定義する．

定義 19（関数呼出しの一般化）$f(k_1,\ldots,k_m,u_{m+1},\ldots,u_n)$ を再帰関数 $f(x_1,\ldots,x_n)$ の 1 つの呼出しとする．ただし，$k_1,\ldots,k_m$ はリストまたは既知の値，そして $u_{m+1},\ldots,u_n$ は未知の値を表す．このとき，式 $f(k_1,\ldots,k_m,x_{m+1},\ldots,x_n)$ を呼出し $f(k_1,\ldots,k_m,u_{m+1},\ldots,u_n)$ の**一般化**と呼ぶ．

定義 19 では，表現しやすくするために，一般性を失うことなく非未知引数はすべて前に揃えるものと仮定していることに注意されたい．部分計算では，新しい剰余関数を生成するために，関数呼出しの一般化が必要である．

定義 20（半剰余条件式）$e_1\oplus e_2$ を 1 つの 2 項関係式とする．e_1' と e_2' がそれぞれ e_1 と e_2 の部分計算された剰余式であるとき，$e_1'\oplus e_2'$ を**半剰余条件式**と呼ぶ．また，2 項関係式 $r_1,\ldots,r_n$ がすべて半剰余条件式であるときに限り，和標準形論理式 $r_1\vee\ldots\vee r_n$ も**半剰余条件式**と呼ぶ．和標準形論理式 $o_1,\ldots,o_n$ がすべて半剰余条件式であるときに限り，積和標準形論理式 $o_1\wedge\ldots\wedge o_n$ も**半剰余条件式**と呼ぶ．

半剰余条件式は条件指標を求めるときに使われる．また，この 2 つの概念は CFC の定義に対して非常に重要である．CFC は，次の 3 種類の部分からなる木構造である：

(1) 根：最初の呼出し．
(2) 節：再帰呼出し節と値節（必ず葉節）の 2 種類がある．そのうち，再帰呼出し節は最初の呼出しに対する部分計算中で出現した再帰呼出し（一般化された再帰呼出しを含む）の節である．値節は計算中で終了条件（再帰条件の反対）が満たされる分岐の終了値の節である．
(3) 辺：ラベル（半剰余条件式）付きの矢印とラベル（一般化 “*gen*”）付きの直線の 2 種類がある．

たとえば，例 1 は 1 つの CFC 例を与える．

例 1（Ackermann 関数）

$$\begin{aligned} a(m,n) \stackrel{\text{def}}{=} [&m{=}\,0\to n+1;\\ &m{\neq}\,0\to[\,n=0\to a(m-1,1);\\ &\quad n{\neq}\,0\to a(m-1,a(m,n-1));]] \end{aligned}$$

$a(2,n)$ を 1 つの呼出しとすれば，$a(2,n)$ の CFC は**図 1** となる．ただし，計算規則は最左最内を仮定する．また，説明を簡単にするために，葉としての再帰呼出しの計算をここでは無視する．

図 1 により，剰余プログラムは次のようになる．

$$a_2(n) \stackrel{\text{def}}{=} [\,n=0\to 3; n\neq 0\to a_1(a_2(n-1));]$$
$$a_1(n) \stackrel{\text{def}}{=} [\,n=0\to 2; n\neq 0\to a_0(a_1(n-1));]$$
$$a_0(n) \stackrel{\text{def}}{=} [\,n+1;]$$

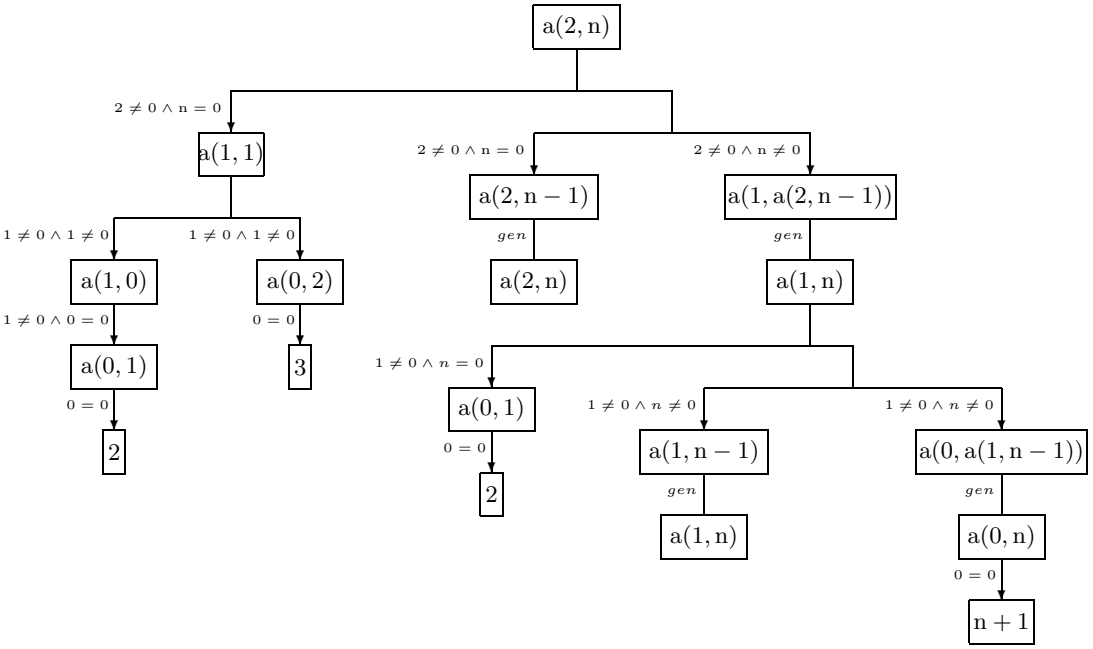
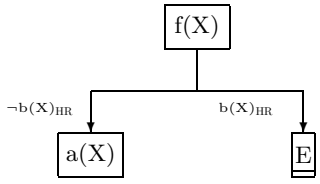


図 1 a(2, n) の CFC．そのうち，矢印左側の式は半剰余条件式，*gen* は一般化を表す．
Fig. 1 The CFC of a(2, n), Where the left of any arrow is half-residual condition expression, *gen* denotes generalization.

(a) $b(X)_R$ = false 時　(b) $b(X)_R$ = true 時
(a) When $b(X)_R$ = false.　(b) When $b(X)_R$ = true.



(c) $b(X)_R$ が未知のとき
(c) When $b(X)_R$ is unknown.

図 2 f(X) の CFC
Fig. 2 The CFC of f(X).

CFC の構成法は，**図 2** と**図 3** により形式的に定義される．任意の再帰関数 f の呼出し f(X) に対して，CFC(f(X)) は図 2 のようになる．そのうち，$b(X)_{HR}$ は論理式 b(X) の半剰余条件式，$b(X)_R$ は論理式 b(X) の剰余式（部分計算の結果）とする．

下線付きの節（たとえば E）は構成変換される必要があるものとする．その構成変換は節に含まれる式の内容によって異なる．具体的には，図 3 のように行われる．

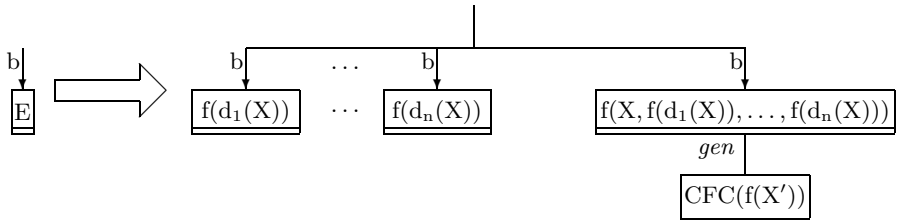
CFC を再帰的に構成する手順は定義 8 の計算意味論に従う．また，根からの任意の計算経路上にラベルとして現れる半剰余条件式の列を**条件式列**，その条件式と対応する条件指標の列を**条件指標列**と呼ぶ．たとえば，図 1 において，根から葉 n + 1 に至る計算経路の条件式列と条件指標列（条件指標関数を H_1 とする）はそれぞれ次の 2 列である：

$2 \neq 0 \wedge n \neq 0$, $1 \neq 0 \wedge n \neq 0$ $0 = 0$
(3, ((“ + 2”), (“U”))), (3, ((“ + 1”), (“U”))), (1, ((“0”)))

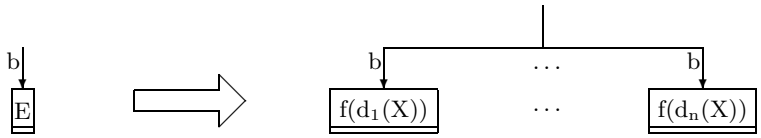
ただし，Ackermann 関数の 3 つの分岐条件式 $m = 0$，$m \neq 0 \wedge n = 0$，$m \neq 0 \wedge n \neq 0$ にはそれぞれ番号 1，2，3 が付けられる．

5.2 CFC と部分計算の停止性

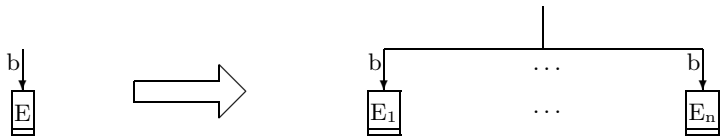
任意の再帰関数呼出しの部分計算は，1 つの CFC に対応する．部分計算中に現れるすべての再帰呼出しは，それぞれこの CFC のノード（値節を除く）に対応する．しかも，節と節の間に 1 つの連結辺（ラベル付きの矢印あるいは直線）がある．CFC の構築中に，ある計算経路の条件指標列に逆順が出現すると，その経路の計算（構成）は停止する．また，条件指標列に逆順が出現するより前に，計算が止まる可能性が次の



(a) $E = f(X, f(d_1(X)), \ldots, f(d_n(X)))$ のとき．ただし，d_i $(1 \leq i \leq n)$ の定義は定義 8 の d_i と同じとする．
(a) When $E = f(X, f(d_1(X)), \ldots, f(d_n(X)))$, where the definitions of d_i $(1 \leq i \leq n)$ is just same as that of d_i in Definition 8.



(b) $E = g(X, f(d_1(X)), \ldots, f(d_n(X)))$ のとき．ただし，d_i $(1 \leq i \leq n)$，g の定義は定義 8 の d_i，g と同じとする．
(b) When $E = g(X, f(d_1(X)), \ldots, f(d_n(X)))$, where the definitions of d_i $(1 \leq i \leq n)$, g are just same as those of d_i, g in Definition 8.



(c) $E = [p \rightarrow E_1; \neg p \rightarrow E_2;]$ のとき．ただし，E_1，E_2 の定義は E と同じとする．
(c) When $E = [p \rightarrow E_1; \neg p \rightarrow E_2;]$, where the definitions of E_1, E_2 are same as that of E.

図 3 E の構成変換
Fig. 3 The transformation of E.

2 つの場合にありうる．

(1) 葉としての節が値節（終了条件を満たすもの）であれば，その葉の所属している経路の計算は止まる．
(2) 葉としての再帰呼出し節がその節の所属している経路にある他の再帰呼出し節とまったく同じ（同じ関数への重複する呼出しなので，畳み込まれる）であれば，その経路の計算は止まる．

この 2 つの処理は自然であるので，ほとんどの部分計算法において同様の処理が行われている．CFC 木構造におけるすべての分岐（経路）の構成が停止すれば，対応する部分計算過程も停止する．すなわち，下記定理 3 のとおり，部分計算の停止性はその CFC 木構造の有限性と等価である．

定理 3 条件指標関数および条件指標上の WQO が与えられているものとする．このとき，図 2 および図 3 で規定される CFC 構成法により作られる任意の再帰関数呼出しに対する部分計算の CFC は，有限の木構造となる（証明は付録 A.1 で述べる）．

6. 部分計算例

再帰条件法とその利点を示すために，本章では，4 つの再帰プログラムを例として再帰条件法による部分計算過程と結果について議論する．

例 2（91 関数）

$$f(n) \stackrel{\text{def}}{=} [\, n > 100 \rightarrow n - 10;\ n \leq 100 \rightarrow f(f(n + 11));\,]$$

再帰呼出し f(98) の CFC は**図 4** の木構造となる．ここで，部分計算は全計算を含むことに注意されたい．

図 4 の任意の計算経路において，条件指標列には逆順がないので，計算は最終の値を求めるまで続く（やり残さない）．実際に，任意の引数 n に対して，再帰条件法による f(n) の計算はやり残さない（証明は付

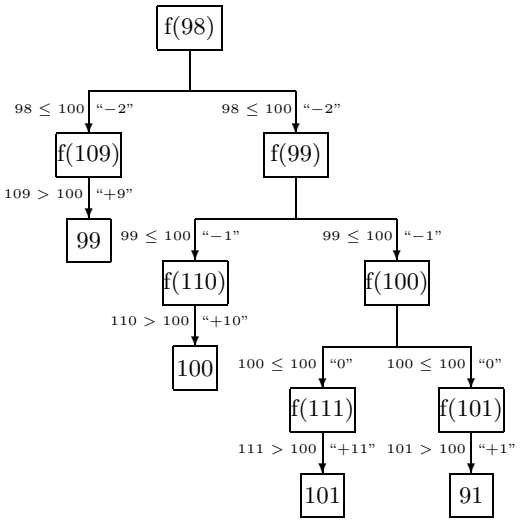


図 4 f(98) の CFC．ただし，矢印の左右はそれぞれ半剰余条件式および条件指標を示す．ここで，条件指標関数と条件指標上の WQO はそれぞれ H と $\preccurlyeq_{\mathrm{emb}}$ である．

Fig. 4 The CFC of f(98), where the left and right of any arrow are half-residual condition expression and condition index respectively. Note condition index function and WQO are H and $\preccurlyeq_{\mathrm{emb}}$ respectively.

録 A.2)．逆に，実引数法は 91 関数の計算では非常にやり残しやすい．たとえば，f(1) を計算する場合に，f(1) = f(f(12)) は明らかである．しかし，文字列 “f(1)” は文字列 “f(12)” に埋め込まれているため，計算は止まって，やり残してしまう．

例 3（Pattern Match 関数）

$$\begin{aligned}\mathrm{M(p,t,w)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{p} = [] \to \mathrm{true};\\ & \mathrm{p} \neq [] \to [\,\mathrm{t} = [] \to \mathrm{false};\\ & \qquad \mathrm{t} \neq [] \to\\ & [\,\mathrm{car(p)} = \mathrm{car(t)} \to\\ & \qquad \mathrm{M(cdr(p), cdr(t), w +\!\!+ [car(p)])};\\ & \mathrm{car(p)} \neq \mathrm{car(t)} \to [\,\mathrm{w} = [] \to \mathrm{M(p, cdr(t), [])};\\ & \qquad \mathrm{w} \neq [] \to \mathrm{M(w +\!\!+ p, cdr(w) +\!\!+ t, [])};]]]]\end{aligned}$$

ただし，x ++y = append(x, y)，変数 p と t はそれぞれパターンとテキストを表す．p = [A, A, B]，w = [] とする．M([A, A, B], t, []) の CFC は**図 5** である．

この例は，部分計算により単純パターンマッチャから KMP パターンマッチャを生成する例として最初に文献 7) で示された．図 5 の任意の計算経路において，条件指標列には逆順は現れない．しかし，再帰呼出し節の葉を持つ経路上にまったく同じ再帰呼出しが出現したので，計算は止まる．したがって，剰余プログラムは次のようになる．

$$\begin{aligned}\mathrm{M_{AAB}(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{A} \neq \mathrm{car(t)} \to \mathrm{M_{AAB}(cdr(t))};\\ & \ \mathrm{A} = \mathrm{car(t)} \to \mathrm{M_{AB}(cdr(t))};]]\end{aligned}$$

$$\begin{aligned}\mathrm{M_{AB}(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{A} \neq \mathrm{car(t)} \to \mathrm{M_{AAB}(t)};\\ & \ \mathrm{A} = \mathrm{car(t)} \to \mathrm{M_{B}(cdr(t))};]]\end{aligned}$$

$$\begin{aligned}\mathrm{M_{B}(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{B} \neq \mathrm{car(t)} \to \mathrm{M_{AB}(t)};\\ & \ \mathrm{B} = \mathrm{car(t)} \to \mathrm{true};]]\end{aligned}$$

実引数法で上述の部分計算をすれば，計算中に現れた呼出し M([A, A, B], [A|t], []) が呼出し M([A, A, B], t, []) を埋め込むため，M([A, A, B], [A|t], []) の計算は止まり，M([A, B], t, [A]) になることができない．したがって，剰余プログラムは次のようになる．

$$\begin{aligned}\mathrm{M_{AAB}(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{A} \neq \mathrm{car(t)} \to \mathrm{M_{AAB}(cdr(t))};\\ & \ \mathrm{A} = \mathrm{car(t)} \to \mathrm{M_{AB}(cdr(t))};]]\end{aligned}$$

$$\begin{aligned}\mathrm{M_{AB}(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{A} \neq \mathrm{car(t)} \to \mathrm{M_{AAB}(t)};\\ & \ \mathrm{A} = \mathrm{car(t)} \to \mathrm{M_{B}(cdr(t))};]]\end{aligned}$$

$$\begin{aligned}\mathrm{MB(t)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{t} = [] \to \mathrm{false};\ \mathrm{t} \neq [] \to\\ & [\,\mathrm{B} \neq \mathrm{car(t)} \to \mathrm{M_{AAB}([A|t])};\\ & \ \mathrm{B} = \mathrm{car(t)} \to \mathrm{true};]]\end{aligned}$$

上述の 2 つの剰余プログラムを比べると，明らかに前者は後者より優れている．特に，パターン p の長さが長ければ長いほど（たとえば，p = [A, A, A, A, B, B, B, B]）その差が拡大する．

例 4（Hailstorm 関数の有限領域への縮小 h(x, y)）

$$\begin{aligned}\mathrm{h(x,y)} \stackrel{\mathrm{def}}{=} [\ & \mathrm{x} > 10^6 \vee \mathrm{x} < -10^6 \vee \mathrm{x} = 1 \to \mathrm{y};\\ & \mathrm{x} \leq 10^6 \wedge \mathrm{x} \geq -10^6 \wedge \mathrm{x} \neq 1 \to\\ & [\,\mathrm{odd(x)} \to \mathrm{h}(3\mathrm{x}+1, \mathrm{y}-1);\\ & \ \neg\mathrm{odd(x)} \to \mathrm{h}(\mathrm{x}/2, \mathrm{y}-1);]]\end{aligned}$$

任意の x の値に関する再帰条件法による h(x, y) の部分計算にはやり残しがなく，結果は y − K（K は計算中で関数 h が呼び出された回数）となる．これは $\mathrm{x} > 10^6 \vee \mathrm{x} < -10^6 \vee \mathrm{x} = 1$ のときは明らか．また，$\mathrm{x} \leq 10^6 \wedge \mathrm{x} \geq -10^6 \wedge \mathrm{x} \neq 1$ のときは，x のすべての値（200 万個）について実際に部分計算して確認できる．

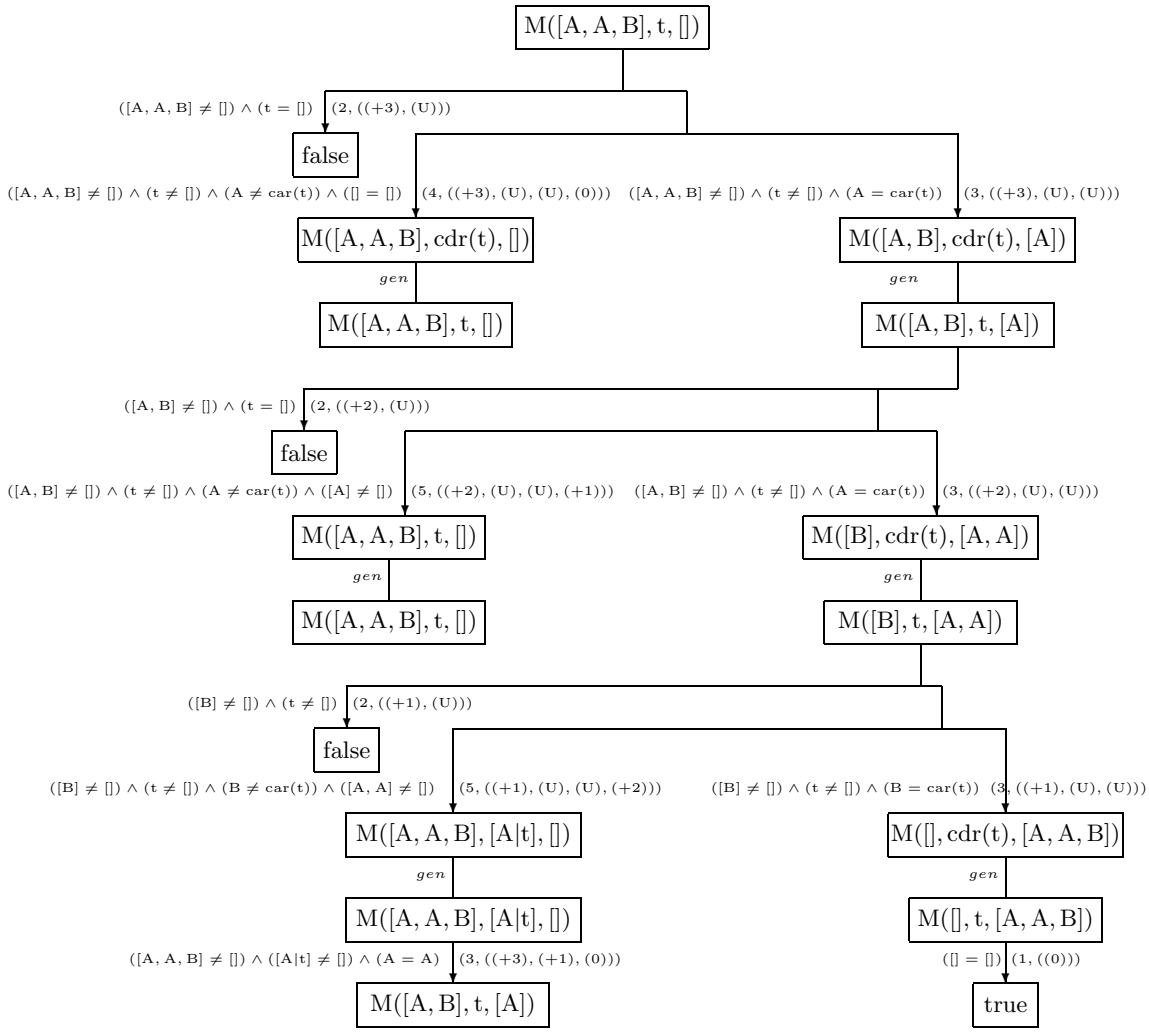


図 5 M([A, A, B], t, []) の CFC. ただし，条件指標関数と WQO はそれぞれ H_1 と $\precsim_1$ である．また，5 つの分岐条件式 p = [], p ≠ [] ∧ t = [], p ≠ [] ∧ t ≠ [] ∧ car(p) = car(t), p ≠ [] ∧ t ≠ [] ∧ car(p) ≠ car(t) ∧ w = [], p ≠ [] ∧ t ≠ [] ∧ car(p) ≠ car(t) ∧ w ≠ [] はそれぞれ番号 1, 2, 3, 4, 5 を付けられる．

Fig. 5 The CFC of M([A, A, B], t, []), where condition index function and WQO are H_1 and $\precsim_1$ respectively. Note that the five branch conditions p = [], p ≠ [] ∧ t = [], p ≠ [] ∧ t ≠ [] ∧ car(p) = car(t), p ≠ [] ∧ t ≠ [] ∧ car(p) ≠ car(t) ∧ w = [], p ≠ [] ∧ t ≠ [] ∧ car(p) ≠ car(t) ∧ w ≠ [] are numbered 1, 2, 3, 4 and 5 respectively.

一方，実引数法に基づいて h(x, y) を x について部分計算すると，やり残しやすい．たとえば，h(7, y) の部分計算では，$h(7, y) = h(22, y-1) = h(11, y-2) = h(34, y-3) = h(17, y-4)$，文字列 “h(7,y)” は文字列 “h(17,y-4)” に埋め込まれているため，計算は止まり，やり残しが生じる．

次には，引数 x を未知，y を既知の場合について考える．このとき，部分計算を続けてもプログラムの効率化が図られないことは明らかである．再帰条件法では再帰条件におけるすべての 2 項関係式が未知なので，部分計算がただちに止まる．しかし，実引数法によると，引数上の逆順がない限り計算は続き，無意味な剰余コードを生成する．たとえば，h(x, 100) は $h(3x+1, 99)$ と $h(x/2, 99)$ を呼び出す．また $h(3x+1, 99)$ は $h(9x+3, 98)$ と $h(3x+1/2, 98)$ を呼び出し，さらに $h(x/2, 99)$ は $h(3x/2+1, 98)$ と $h(x/4, 98)$ を呼び出す．この再帰呼出しの過程はさらに続くが，それは剰余プログラムの性能を良くするこ

となしに，無駄にサイズを大きくするだけである．上述のように，再帰関数に与えられた既知の引数が再帰条件の値に影響を与えない場合に，実引数法はやりすぎやすい．しかし，再帰条件法にもやり残しとやりすぎがある．特に，再帰条件が 2 項関係式 $e_1 = e_2$ である場合に，再帰条件法でやり残しが発生する場合が多い．

例 5（再帰条件法でやり残す例）

$$g(m,n) \stackrel{\text{def}}{=} [\ mod(m,n) \neq 0 \to 1;\ mod(m,n) = 0 \to g(m/n, n+1) + 1;\ ]$$

実引数 m と n がともに既知の場合でさえも g(m, n) の計算に対して，再帰条件法はやり残す場合がある（たとえば m = 12, n = 2 の場合）．

7. 例による再帰条件法と実引数法の比較

本章では，より複雑な例を用いて本稿の提案する再帰条件法と従来の実引数法との比較を行う．

例 6 再帰関数 C を次のように定義する：

$$C(x,y,z) \stackrel{\text{def}}{=} [\ x = y \to [\, y = z \to [\,];\ y \neq z \to [z \,|\, C(x,y,z+1)];\,]\ x \neq y \to [x \,|\, C(x+1,y,z)];\,]$$

ただし，$x,y,z \geq 0$ かつ $y \geq x,z$．C は x と y の間の正整数および z と y の間の正整数を要素とするリストを求める関数である．

以下では，関数 C の既知引数を 3 つのケースに分けて，再帰条件法と実引数法に基づく部分計算の効果を議論する．ただし，詳細な証明は紙面の都合で省略する．ここで，再帰呼出し $C(x+1,y,z)$ と $C(x,y,z+1)$ では x と z の値が計算に従って増えることに注意されたい．

(1) 引数 x と y あるいは z と y が既知の場合．

実引数法よりも再帰条件法の方が，やり残しおよびやりすぎともに少ない．たとえば，$C(0,11,z)$ を 1 つの呼出しとする．再帰条件法による部分計算では，唯一の再帰計算経路は $C(0,11,z), C(1,11,z), \cdots, C(11,11,z), C(11,11,z+1)$ である．この計算経路の条件式列は $(0 \neq 11), (1 \neq 11), \cdots, (10 \neq 11), (11 = 11) \wedge (11 \neq z)$，条件指標列（分岐条件の番号は省略する）は $((\text{“}-11\text{”})), ((\text{“}-10\text{”})), \cdots, ((\text{“0”})), ((\text{“0”}), (\text{“U”}))$ である．条件指標列には逆順がないが，呼出し $C(11,11,z+1)$ が $C(11,11,z)$ のように一般化されるため，計算が止まる．剰余プログラムは：

$$C_{(0,11)}(z) \stackrel{\text{def}}{=} [0,1,2,3,4,5,6,7,8,9,10 \,|\, C_{(11,11)}(z)]$$

$$C_{(11,11)}(z) \stackrel{\text{def}}{=} [\ 11 = z \to [\,];\ 11 \neq z \to [z \,|\, C_{(11,11)}(z+1)];\,]$$

これは呼出し $C(0,11,z)$ の剰余プログラムとしては能率が良い（すなわち，やり残しもやりすぎも少ない）．実引数法による部分計算では，呼出し $C(0,11,z)$ が $C(10,11,z)$ に埋め込まれるため，計算が呼出し $C(10,11,z)$ で止まり，下記の能率の悪い剰余プログラムが生成される：

$$C_{(0,11)}(z) \stackrel{\text{def}}{=} [0,1,2,3,4,5,6,7,8,9 \,|\, C(10,11,z)]$$

$$C(x,y,z) \stackrel{\text{def}}{=} [\ x = y \to [\, y = z \to [\,];\ y \neq z \to [z \,|\, C(x,y,z+1)];\,]\ x \neq y \to [x \,|\, C(x+1,y,z)];\,]$$

(2) 引数 x，y および z がすべて既知の場合．

本質的にケース (1) と同じであり，再帰条件法に基づく計算は関数の最後の値を求めるまで続き，やり残さない．

(3) x と z が既知あるいは x，y または z のうちの 1 つが既知の場合．

再帰条件法によれば，関数 C に含まれる分岐条件においてすべての 2 項関係式が未知であるため条件指標上の逆順がすぐ出現し，計算は止まる．原始プログラムとほぼ同じ剰余プログラムが得られる（停止しなければ，効率の悪い冗長な剰余コードが生成される）．逆に，実引数法によると，実引数上の逆順が現れるまで計算が続くため，やりすぎが起こりやすい．

ここで，関数 C を下記のように定義し直す：

$$C(x,y,z) \stackrel{\text{def}}{=} [\ x = y \to [\, y = z \to [\,];\ y \neq z \to [z \,|\, C(x,y,z-1)];\,]\ x \neq y \to [x \,|\, C(x-1,y,z)];\,]$$

ただし，$x,y,z \geq 0$ かつ $y \leq x,z$．今度は，C は y と x の間の正整数および y と z の間の正整数を要素とするリストを求める関数となる．

ケース (1) と (2) では，実引数法および再帰条件法に基づく $C(x,y,z)$ の部分計算はともに能率の良い剰余プログラムを生成する（実引数 x あるいは z の値は単調減少する．また，条件指標 $H(x \neq y)$ および $H(y \neq z)$ も “単調減少” する）．一方，ケース (3) では実引数法はやりすぎやすい．

両方法で数多くの再帰プログラムを部分計算してみた結果，再帰条件法の方が実引数法よりも停止のタイ

ミングが合理的な場合が多いという感触を我々は得ている．しかし，実際には，再帰条件法よりも実引数法の方がやり残しとやりすぎの少ない場合もある．

8. 相互再帰プログラムへの拡張

議論を簡単にするために，本稿では単一再帰関数だけを取り上げた．本稿における議論は，下記のようにすれば相互再帰関数にも適用可能である．

相互再帰関数の場合に，逆順を出にくくするためには，異なる関数の条件指標どうしは比較できないことが望ましい．そのため，各分岐条件に通し番号を付けた場合と同様にして，各関数に唯一の番号を付ける．そして対（番号，元の条件指標）のように元の条件指標を一次元拡大し，かつ $\precsim_1$ の直積空間も一次元拡大する．そうすると，同じ計算経路の条件指標列に逆順があれば，それは必ず同じ関数の条件指標どうし間の逆順となる．

9. 混 合 法

前章までに調べたとおり，やり残しに関しても，再帰条件法も実引数法もともに問題に応じて得手不得手がある．しかし実引数法は我々が提案した再帰条件法と同じく同相埋め込み関係に基づく停止条件であるので，両者を組み合わせ，より停止のタイミングの遅い決定可能な停止性判定法を作ることができる．すなわち，再帰条件と実引数を対にして新しい再帰条件を作り，対の 2 つの要素がともに逆順になるまで計算を続ければよい．

まず，条件指標を次のように拡張する．再帰関数定義に含まれる再帰条件 b(X) において，X は実引数であるとする．その X を 1 つの次元として条件指標に追加する．その際，条件指標に使われる文字のアルファベット α は言語に使われるすべての文字の集合とする．この集合が有限集合であれば，前に定義されたすべての同相埋め込み関係は依然 WQO になる．たとえば，新しい条件指標関数 H_2 同相埋め込み関係 $\precsim_2$ を次のように定義することができる．

定義 21 （条件指標関数 H_2）H_2：論理式 $\to \alpha^* \times(\mathrm{Int} \times \bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m)$，任意の条件式 b(X) に対して，

$$H_2(b(X)) \stackrel{\mathrm{def}}{=} (X, H_1(b(X)))$$

定義 22 （擬順序 $\precsim_2$）$\alpha^* \times (\mathrm{Int} \times \bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m)$ 上の 2 項関係 $\precsim_2$ を α^* 上の埋め込み関係 $\preccurlyeq_{\mathrm{emb}}$ と $\mathrm{Int} \times \bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の埋め込み関係 $\precsim_1$ により次のように定義する：

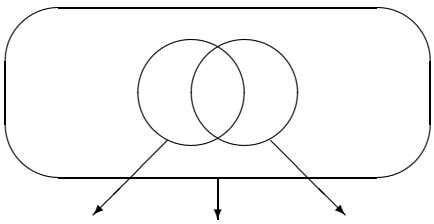


図 6 やり残しに関する実引数法，再帰条件法および混合法の比較．ただし，各楕円はやり残しのない部分計算の集合（関数名と既知変数およびその値の集合）を表す．

Fig. 6 The comparison and relation of APM, RCM and CM in the sense of omission error, where any ellipse indicates the set of partial evaluations which can be sufficiently computed if the corresponding terminating technique is used.

$$(a_1, a_2) \precsim_2 (b_1, b_2) \text{ iff } a_1 \preccurlyeq_{\mathrm{emb}} b_1 \wedge a_2 \precsim_1 b_2$$

系 1 により，$\precsim_2$ は $\alpha^* \times (\mathrm{Int} \times \bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m)$ 上の WQO である．混合法では，逆順があれば，必ず α^* 上の逆順と $\mathrm{Int} \times \bigcup_{m=1}^{M}(\bigcup_{n=1}^{N}(\alpha^*)^n)^m$ 上の逆順は同時に現れるので，実引数法と再帰条件法の両者よりも，やり残しが少なくなる．実引数法，再帰条件法および混合法のやり残しに関する比較を**図 6** に示す．

また，**表 1** は 10 個の原始プログラムに対して，3 つの停止性判定法に基づく部分計算効果の比較を与える．ただし，71 関数[7]，Hailstorm 関数，Merge および竹内関数の定義については付録 A.3 を参照されたい．

表 1 のとおり，混合法はやり残しを大きく減らすことができる．ただし，やり残しを減らす一方でやりすぎを増やす可能性もある．やりすぎを恐れずに剰余プログラムの効率を追求する場合には，混合法が有効である．

10. 終 わ り に

部分計算のための新しい停止性判定法として再帰条件法を提案し，従来の実引数法との利害得失について詳しい比較を行った．再帰条件法は，実引数の代わりに，再帰プログラムにおける再帰呼出しの条件を利用する．そのため，再帰条件法は，実引数法よりも再帰呼出しに敏感となり，最適化エラーの小さい剰余プログラムの生成を可能とする場合が多い．さらに，再帰条件法と実引数法の両者の停止法を利用した混合法を提案した．混合法は，WQO を用いて部分計算を必ず止めることのできる既存の判定法の中では，つねに一番遅く止まることのできるものである．部分計算の決定可能な停止条件に関して，実用的観点も含めて行われた議論は本稿が最初であると我々は考えている．再帰条件法では，論理式に含まれる 2 項関係式が最も重

表 1 実引数法，再帰条件法および混合法による部分計算効果の比較．ただし，条：再帰条件法，実：実引数法，混：混合法，$\succ$：より多い，$=$：同じ，$\not\prec\not\succ$：量的に比べられない．
Table 1 The comparison of partial evaluation effectiveness based on recursion condition method and actual parameter method and combined method, where, $\prec$ indicates more than, $=$ indicates equal and $\not\prec\not\succ$ indicates incomparable.

原始プログラム	既知引数	効果					
		やり残しの有無			やり残し多少の比較	やりすぎの有無			やりすぎ多少の比較
		条	実	混		条	実	混	
91 関数	n	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
71 関数	n	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
Hailstorm 関数	n	有	有	有	条 $\not\prec\not\succ$ 実，条 $\succ$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
M 関数（例 3）	p, t	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
	p	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\prec$ 混	無	無	無	条 $=$ 実 $=$ 混
	t	無	無	無	条 $=$ 実 $=$ 混	無	有	有	実 $\succ$ 条，混 $\succ$ 条，実 $=$ 混
Ackermann 関数	m, n	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
	m or n	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
H 関数（例 4）	x, y	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	y	無	無	無	条 $=$ 実 $=$ 混	無	有	有	実 $\succ$ 条，混 $\succ$ 条，実 $=$ 混
G 関数（例 5）	m, n	有	無	無	条 $\succ$ 実，実 $=$ 混，条 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	m or n	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
Merge 関数	x, y	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
	x or y	無	無	無	条 $=$ 実 $=$ 混	無	無	無	条 $=$ 実 $=$ 混
C 関数（例 6）	x, y, z	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x, y	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	y, z	無	有	無	実 $\succ$ 条，条 $=$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x, z	無	無	無	条 $=$ 実 $=$ 混	無	有	有	実 $\succ$ 条，混 $\succ$ 条，実 $=$ 混
	x or y or z	無	無	無	条 $=$ 実 $=$ 混	無	有	有	実 $\succ$ 条，混 $\succ$ 条，実 $=$ 混
竹内関数 t	x, y, z	有	有	有	条 $\not\prec\not\succ$ 実，条 $\succ$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x, y	有	有	有	条 $\not\prec\not\succ$ 実，条 $\succ$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	y, z	有	有	有	条 $\not\prec\not\succ$ 実，条 $\succ$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x, z	有	有	有	条 $\not\prec\not\succ$ 実，条 $\succ$ 混，実 $\succ$ 混	無	無	無	条 $=$ 実 $=$ 混
	x or y or z	無	無	無	条 $=$ 実 $=$ 混	無	有	有	実 $\succ$ 条，混 $\succ$ 条，実 $=$ 混

要である．任意の n ($n \geq 1$) 項関係式に対して，2 項関係式に変換せずに有効な条件指標関数を見つけることができればやり残しをさらに減らすことが期待できる．また，再帰条件としての 2 項関係式の 2 項関係 $\oplus$ が $=$ になる場合に，現在の条件指標はやり残しを起こしやすい．したがって，この問題を解決できる条件指標関数を見つけることができれば，再帰条件法をさらに合理的かつ実用的にすることが可能である．それらの発見法の確立が今後の課題である．

参 考 文 献

1) Andersen, L.: Program analysis and specialization for the C programming language, Ph.D. Thesis, DIKU, University of Copenhagen (1994).
2) Bondorf, A. and Danvy, O.: Automatic autoprojection of recursive equations with global variables and abstract data types, *Science of Computer Programming*, Vol.16, pp.151–195 (1991).
3) Bruynooghe, M., Schreye, D.D. and Martens, B.: A general criterion for avoiding infinite unfolding during partial deduction, *New Generation Computing*, Vol.11, No.1, pp.47–49 (1992).
4) Consel, C., et al.: A uniform approach for compile-time and run-time specialization, *Partial Evaluation*, Danvy, O., Glück, R. and Thiemann, P. (Eds.), LNCS, Vol.1110, pp.54–72, Springer-Verlag (1996).
5) Dershowitz, N.: Termination of rewriting, *Journal of Symbolic Computation*, No.3, pp.69–116 (1987).
6) 二村良彦，小西善二郎：一般部分計算法に基づく自動プログラム変換実験システムの開発，コンピュータソフトウェア学会論文誌（印刷中）．
7) Futamura, Y., Nogi, K. and Takano., A.: Essence of generalized partial computation, *Theoretical Computer Science*, No.90, pp.61–79 (1991).
8) 二村良彦，Song, L.，小西善二郎：一般部分計算（GPC）における制御構造と停止条件，日本ソフトウェア科学会第 15 回大会論文集，D5-1, pp.313–316 (1998).
9) Glück, R. and Sorensen, M.: A roadmap to

supercompilation, *Proc. 1996 Dagstuhl Seminar on Partial Evaluation*, Danvy, O., Glück, R. and Thiemann, P. (Eds.), LNCS, Vol.1110, pp.137–160, Springer-Verlag (1996).
10) Higman, G.: Ordering by divisibility in abstract algebras, *Proc. London Mathematical Society*, Vol.2, pp.326–336 (1952).
11) Kruskal, J.: Well-quasi ordering, the tree theorem, and Vazsonyi's conjecture, *Trans. American Mathematical Society*, No.95, pp.210–225 (1960).
12) Barwise, J.: *Handbook of mathematical logic*, North-Holland (1977).
13) Leuschel, M.: On the power of homeomorphic embedding for online termination, *SAS'98*, pp.230–245 (1998).
14) Leuschel, M., Martens, B. and Schreye, D.D.: Controlling generalization and polyvariance in partial deduction of normal logic programs, *TOPLAS*, Vol.20, No.1, pp.209–258 (1998).
15) Martens, B. and Schreye, D.D.: Automatic finite unfolding using well-founded measures, *The Journal of Logic Programming*, Vol.28, No.2, pp.89–146 (1996).
16) Middeldorp, A. and Zantema, H.: Simple termination of rewrite systems, *Theoretical Computer Science*, Vol.175, No.1, pp.127–158 (1997).
17) Nash-Williams, C.: On well-quasi-ordering finite trees, *Proc. Cambridge Philos. Soc*, Vol.59, No.4, pp.833–835 (1963).
18) Ruf, E.: Topics in online partial evaluation, Ph.D. Thesis, Standford University (1993).
19) Sorensen, M. and Glück, R.: An algorithm of generalization in positive super-compilation, *Proc. ILOS'95*, Portland, USA, Lloyd, J.W. (Ed.), pp.465–479 (1995).
20) Sorensen, M., Glück, R. and Jones, N.: A positive supercompiler, *Journal of Functional Programming*, Vol.6, No.6, pp.811–838 (1996).
21) Turchin, V.: The concept of a supercompiler, *TOPLAS*, Vol.8, No.3, pp.292–325 (1986).
22) Weise, D., Conybeare, R., Ruf, E. and Seligman, S.: Automatic online partial evaluation, *Proc. Conference on Functional Programming Languages and Computer Architectures*, LNCS, Vol.523, pp.165–191, Harvard University, Springer-Verlag (1991).

付　　録

A.1 定理 3 の証明

CFC の定義により，CFC の木構造において，節は再帰呼出し節と値節（必ず葉節）の 2 種類ある．辺はラベル（半剰余条件式）付きの矢印とラベル（“*gen*”）付きの直線の 2 種類ある．また，上下に隣接する節と節の間には辺は 1 つしか存在しない．CFC の定義により，任意の節に対して，子供節の個数はたかだか関数定義における再帰呼出しを含まない分岐の数とすべての再帰呼出しの個数の和である．したがって，CFC は有限分岐の木構造である．

補題 1 （Konig）有限分岐の無限木は必ず無限な経路を持つ（証明は文献 12) を参照されたい）．

CFC の定義により，任意の 2 つの再帰呼出しの間に必ず 1 つの条件式あるいは “*gen*” が付いている．また，根から始まる任意の経路上では，半剰余条件式の付いている辺の数が “*gen*” の付いている辺の数より 1 本多い．しかも，根から始まる任意の経路上にあるラベルとしての条件式（葉の直前の条件式を除けば）の条件指標列は，必ずある WQO に関する有限な許容列である．したがって，経路上の辺数は有限である．すべての経路が有限なので，上記補題と背理法により，CFC は有限な木構造である．（QED）

A.2 再帰条件法で 91 関数の計算がやり残さないことの証明

91 関数を f(n) とする．この関数の値が $n > 100$ のとき $n - 10$，そして $n \leq 100$ のときに 91 となる事実を利用して，構造帰納法を用いて証明する．すなわち，f(n) の定義域，すなわち任意の整数 n についてある性質 P(n) が成立することの証明を下記の 2 段階により行う．$k \leq 100$ なるある整数 k に対して，

(1) $n > k$ のとき，P(n) が成立する．
(2) $n \leq k$ のとき，P(n + 11) の成立を仮定すれば P(n) が成立する．

まず次の補題を上記の構造帰納法により示す．このときの k は 78 である．

補題 2 $n \leq 89$ ならば，f(n) の計算中で出現する f への再帰呼出しの実引数の値は n より大きい．

証明：$n + 11 \leq 100$ なので，f の定義より，$f(n) = f(f(n + 11)) = f(91)$．したがって，f(n) の計算は，まず f(n + 11) の計算を行い，次に f(91) の計算を行うことが分かる．f(91) の計算に現れる f への再帰呼出しは次のとおりである．したがって，その実引数はどれも 91 より大きい．

$$f(91) = f(f(102)) = f(92) = \ldots = f(101) = 91$$

さらに，f(n + 11) の計算に現れる再帰呼出しの実引数は n より大きいことを示す．

(1) $n > 78$ のとき，すなわち $79 \leq n \leq 89$ ならば，f(n + 11) の計算に現れる再帰呼出しは次のようになり，その実引数はどれも n+11 より大きい．

$$f(n+11) = f(f(n+22)) = f(n+12) = \ldots = f(101) = 91$$

(2) $n \leq 78$ のとき，$n+11 \leq 89$ であるので，$f(n+11)$ の計算に現れる再帰呼出しの実引数は $n+11$ より大きいと仮定する（構造帰納法の仮定）．すると，これらの実引数は当然 n より大きい．(QED)

次には $k = 89$ として下記の定理を証明する．

定理 4 任意の整数 n に対して，再帰条件法で 91 関数の計算はやり残さない．

証明：

(1) $n > 100$ ならばすぐ止まるので明らか．$90 \leq n \leq 100$ のとき，$90, \ldots, 100$ の合計 11 個の引数に対して実際に計算してみればやり残しのないことが確かめられる．

(2) $n \leq 89$ のとき，$f(n+11)$ の計算はやり残しがないと仮定する（構造帰納法の仮定）．$n \leq 100$ の再帰条件指標は $t(n-100)$ であり，その CFC において根 $\boxed{f(n)}$ は 2 人の子供 $\boxed{f(n+11)}$ と $\boxed{f(91)}$ を持つ．構造帰納法の仮定より，CFC($f(n+11)$) には，葉の直前の辺の条件数を除けば，逆順はない．また，CFC(91) は実際に調べてみれば，葉の直前の辺を除けば，逆順がないことが分かる．一方，$n-100 < 0$ であるので，これは子孫の正のラベル（正確には，ラベルの条件指数）とは逆順にならない（正および負のラベルは各々'+' および '−' で始まるからである）．また，補題より子孫に負のラベルがあれば，それは $n-100$ より大きい．したがって，$t(n-100)$ と逆順になりえない．したがって，$t(n-100)$ と逆順のラベルは，葉の直前のラベルを除けば，CFC($f(n+11)$) および CFC(f(91) にはありえない．したがって，91 関数の計算にはやり残しがない．(QED)

A.3 71，Hailstorm，Merge および竹内関数の定義

71 関数：

$$f71(x) \stackrel{\text{def}}{=} [\, x > 70 \rightarrow x;\ x \leq 70 \rightarrow f71(f71(x+1));\,]$$

Hailstorm 関数：

$$h(x) \stackrel{\text{def}}{=} [\, x = 1 \rightarrow 1;\ x \neq 1 \rightarrow [\, odd(x) \rightarrow h(3x+1);\ \neg odd(x) \rightarrow h(x/2);\,]\,]$$

Merge 関数：

$$M(x,y) \stackrel{\text{def}}{=} [\, x = [] \rightarrow y; x \neq [] \rightarrow [\, y = [] \rightarrow x; y \neq [] \rightarrow [\, x > y \rightarrow [car(y)|M(x, cdr(y))];\ x \leq y \rightarrow [car(x)|M(cdr(x), y)];\,]\,]\,]$$

竹内関数：

$$t(x,y,z) \stackrel{\text{def}}{=} [\, x \leq y \rightarrow y; x > y \rightarrow t(t(x-1,y,z), t(y-1,z,x), t(z-1,x,y));\,]$$

(平成 12 年 3 月 7 日受付)
(平成 12 年 5 月 24 日採録)

宋　立彤（学生会員）

1987 年中国南京大学理工学部計算機科学科卒業．1990 年中国吉林大学大学院計算機科学科修士課程修了．2000 年より吉林大学計算機科学科助教授．現在，早稲田大学大学院理工学研究科博士課程在学中．プログラム変換，部分計算（評価），プログラミングの方法論に興味を持つ．日本ソフトウェア科学会会員．

二村　良彦（正会員）

1965 年北海道大学理工学部数学科卒業．1972 年 Harvard 大学応用数学科大学院修士課程修了．工学博士．1965 年日立製作所入社．日立製作所中央研究所主任研究員および基礎研究所主管研究員を経て，1991 年より早稲田大学理工学部情報学科教授．2000 年より早稲田大学ソフトウェア生産技術研究所所長兼任．その間，Uppsala 大学計算機科学科 Guest Professor（1985–1986）および Harvard 大学計算機科学科 Visiting Scholar（1988–1989）．計算と論理の関係，プログラムの生産性向上，アルゴリズムの教育，設計，解析，評価などに興味を持つ．主要業績：Futamura Projections の発見（1971 年），プログラム技法 PAD の開発と ISO 化，一般部分計算法の発見（1987 年）と米国特許化．日本ソフトウェア科学会，ACM 各会員．

Robert Glück

Robert Glück is an associate professor of Computer Science at the University of Copenhagen. He received his M.Sc. and Ph.D. degrees in 1986 and 1991 from the University of Technology in Vienna, where he also worked as assistant professor. He received his Habiliation (venia docendi) in 1997. He was research assistant at the City University of New York and received twice the Erwin-Schrodinger-Fellowship of the Austrian Science Foundation. Currently he is an Invited Fellow of the Japan Society for the Promotion of Science working at Waseda University in Tokyo. His main research interests are advanced programming languages, theory and practice of program transformation, metaprogramming techniques.