3E−8

再帰除去のごみ回避効果

筧 一彦* 坂本 巨樹* 二村 良彦**

*早稲田大学理工学研究科 **早稲田大学理工学部

1 はじめに

ごみ集め (garbage collection) は, メモリ管理を意識せず安心して自由にプログラミングが行なえる, という利便性を与えてくれる. その利便性故, プログラマがごみの出し方を考慮しなくてよくなったことが, メモリ資源の浪費およびごみ集め時間の増加を招いている. このごみ集めにかかる時間を極力減らすために, メモリ管理およびごみを出す際の工夫の自動化, 即ちごみ回避 (garbage avoidance) の価値が高まってきている.

我々の研究の目的は, プログラマにごみ回避を意識させずに, 言語処理系またはプリプロセッサがそれを自動的に行なう方法の開発である. 我々は現在までに, 再帰プログラムから系統的に再帰を取り除くこと (再帰除去, recursion removal) のごみ回避に関する効果を, 幾つかの Lisp プログラムについて調べた. その結果, 再帰除去は有効なごみ回避法であり, それによりごみ集め時間が大幅に改善され得ることを確認した. 本稿ではその調査結果について報告する.

2 ごみ回避

ごみ回避の問題に触れる前に, まずメモリ管理に関連する 3 つのプロセスを明らかにする:

- 利用可能なメモリを自由セルリスト (ヒープ) およびスタックとして管理し, プログラムからの要望に応じて割り当てるメモリセル (記憶場所) 割当器 (SA);
- SA から割り当てられたセルを使い捨てるプログラム (PG);
- プログラムが使い捨てた不要なセルをごみとして回収し, 再生利用できるように自由セルリスト (またはスタック領域) に戻すごみ集め器 (GC).

これを踏まえたうえで, ごみ集め時間を短縮するために, ごみを出す際の操作として次のようなものが考えられる.

廃物利用 不要となったセルをごみとして出さずに, 新たなデータのために利用する. PG が SA を呼び出さずに自分でメモリ割り付けを行なうので, PG の性能も向上し, またごみの発生が押さえられる.

寿命短縮 PG で必要なデータの寿命を短くする. これによってセルの循環が早まることになる. 例えば, 関数の実引数をスタックに配置したり (スタック配置). あるいは, PG が自分で不要と判断したらその時点でそのデータを切り捨てることなどが, その実現法となる.

個別回収 ごみが出たときに, GC を経由せずに SA に渡し, 自由セルリストに直接戻す. これによって, GC の起動を回避できる.

スタック排除 PG において不要なスタックを積むことを避ける. スタック自身もメモリ資源であると同時に, そこに入っているポインタからの参照でセルの生存を判断するため, 不要にスタックを積むと, ごみ集めの際に大きな無駄が発生することになる. PG における関数呼び出しをインライン展開しスタックを不要とする操作がこの一例である.

本稿では, 上記の各操作をごみ回避と呼ぶ. まず, ごみ集めに掛かる総時間とごみ回避との関係について簡単に説明する. 参照カウント方式 (reference counting) 以外のごみ集め方式で, それにかかる総時間を決める要因は, 1 回のごみ集めにかかる時間 (回収時間) とごみ集めの頻度 (回収頻度) とである. 回収時間は, ごみ集め時に生存しているセルの数 (同時使用量) が多いほどその時間が多くなる. 回収頻度は, メモリ使用の総量 (延べ使用量) およびメモリの同時使用量の両方が多いほど回数が増える. 従ってごみ回避操作は, 少なくともどちらか一方の使用量を減らす必要がある. 表 1 に, 上記のごみ回避操作の期待される効果を示した.

Table 1: ごみ回避法の期待される効果

ごみ回避操作	削減対象	
	延べ使用量	同時使用量
廃物利用	○	○
寿命短縮	—	○
個別回収	—	○
スタック排除	○	○

[3] を初めとしてコンパイル時ごみ集め (compile-time garbage collection) として説明されているものは, 廃物

An Evaluation of Recursion Removal in terms of Garbage Avoidance.

KaZuhiko KAKEHI*, Naoki SAKAKMOTO* and Yoshihiko FUTAMURA**.

*Graduate School of Science and Engineering, Waseda University

**School of Science and Engineering, Waseda University

Table 2: マージソートの実行時間

		speed=3, space=0			speed=0, space=3			セル数
		total	(non-gc)	(gc)	total	(non-gc)	(gc)	
top-down	再帰版	294,790	130,330	164,460	327,950	137,010	190,940	236,080
	反復使用版	130,380	82,850	47,530	168,920	109,790	59,130	236,079
	差	164,410	47,480	116,930	159,030	27,220	131,810	1
	改善率	0.442	0.636	0.289	0.515	0.801	0.310	1.000
bottom-up	再帰版	129,190	81,630	47,560	139,240	84,860	54,380	97,060
	反復版	57,900	40,460	17,440	95,030	45,030	50,000	97,060
	差	71,220	41,170	30,120	44,210	39,830	4,380	0
	改善率	0.448	0.496	0.367	0.682	0.531	0.919	1.000

セル数は $\times 10^3$, それ以外の単位は msec.
差 = (再帰版) − (反復版) とし, 改善率 = (反復版) / (再帰版) とする. よって, 改善率の値が小さいほど良いことになる.

利用や個別回収に相当するものである. また, これらとは別に, 延べ使用量を直接減少させる方法として融合法 (deforestation, fusion) [4] がある. こうしたごみ回避手法は基本的に独立に作用するもののため, オーバーヘッドが上回ってしまうまでは相乗効果が期待できる.

3 再帰除去

再帰除去とは, 与えられた再帰プログラムを, スタックを用いずにしかも計算量を増加させずに反復プログラムに変換することをいう.

再帰除去は関数の状態を変換するもので, データ生成の状態に変化を及ぼすものでは基本的にない. しかし, 再帰除去でスタックに積む関数の数を大幅に減らすことで, 表1でのスタック排除の効果が期待できる. また, スタックに余裕が生じるので, 再帰除去後のプログラムに工夫をすることで, 寿命短縮の効果も生むことができる.

4 実験

この再帰除去のごみ回避効果を測定するために, Sun Ultra Enterprise 2 上で Allegro Common Lisp (ACL) を用いて実験を行った. プログラムにはマージソートを使い, トップダウンに行うものとボトムアップに行うものとで, それぞれ再帰的に書かれたものと, それらから [1] で紹介されている手法を用いて再帰除去を行なったものとでのデータを取った. この手法の自動化は, [2] で試みられている.

トップダウンのものは, 与えられたリストを 2 分割 (halve) したものをマージ (merge) していくもので, この 2 つの関数からは再帰除去が可能だが, それを呼び出すおおもとの関数 mergesort は線形再帰ではないため再帰除去が施されていない. ボトムアップのものは, 隣り合う 2 つの要素を merge して 1 本のリストを作り, これを繰り返すことでソートを行う. このソートを構成する関数それぞれからの再帰除去が可能である.

トップダウンとボトムアップ, 再帰版と再帰除去版, オプション speed, space, safety, debug を speed のみ3にして他を0にしたものと space のみ3にして他を0にしたもの, の合計 8 種類に対して, 自由度 100,000, 長さ 60,000 のリストを 100 回ソートさせた時間を測定した. その結果と比較, そして ACL で測定できる cons セルの数をまとめたものが表 2 である.

この表 2 からわかるように, 全体での実行時間はほぼ半分に, ごみ集めにかかる時間はほとんどのもので 3 割程度に抑えられている. 今回の実験ではデータの性質からスタックにデータを積ませておらず, cons セルの数も再帰と再帰除去とで変化がないので, 再帰除去の回収時間短縮の効果によるところが大きいものと思われる.

5 おわりに

ごみ回避操作とその持つ意味を分類し, 再帰除去がそのうちのどれに当てはまるのかを明らかにした. そのうえで, 実際にどれだけの効果をあげるのか実験を行った.

実験でも示されているように, 再帰除去の持つごみ回避効果は高い. 加えて, 他のごみ回避手法やごみ集めの手法から基本的に独立であるという利点がある. 今後はより多くのごみ回避操作および手法を調べ, それらの相互関係や適用範囲について明らかにしたい.

References

[1] 二村, 大谷. 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. 1998 年.

[2] 坂本, 川本, 小西, 二村. 線形再帰プログラムからの再帰除去法の実現, 情報処理学会第 56 回全国大会論文集, 3E-06, 1998 年 3 月.

[3] J.M. Barth. Shifting Garbage Collection Overhead to Compile Time. In *Communication of the ACM*, 20(7): 513–518. July 1977.

[4] P.L. Wadler. Deforestation: Transformating programs to eliminate trees. In *European Symposium on Programming*, volume 300 in Lecture Notes in Computer Science. Springer-Verlag, 1988.