2E-4 Ｌｉｓｐプログラムの静的解析によるＧＣの改良

五味　弘　西垣　秀樹　長坂　篤
(株)沖テクノシステムズラボラトリ　沖電気工業株式会社

１．はじめに

いままでに提案されたＧＣのアルゴリズムは個々のプログラムの特性とは無関係であるのがほとんどであった。そこで我々はＬｉｓｐプログラムの静的解析を行いあらかじめガーベジとなる領域を見つけ、そのことを利用することによりＧＣの改良を行う。

ＧＣには一括型ＧＣとリアルタイム型ＧＣがあるが、本研究では一括型ＧＣを対象とした。

Ｌｉｓｐプログラムの静的解析によるＧＣの改良はいくつか報告されているが[(1)-(2)]、我々は通常のCommon Lispを対象にし、その上で効率の良いＧＣを目的とした。

２．ガーベジの生成過程

まず静的解析を行うために必要となるガーベジの生成する過程の分析を行った。文献(1)ではいくつかのプログラムにおいてガーベジの生成過程を実験的に分析している。我々はその結果と定性的な分析によりガーベジの生成する主な過程を以下の４種類に分類した。

TYPE 1:setqなどの副作用のある関数によるガーベジの生成

setqなどによって新たにポインタの付けかえが行われるとき，以前に指されていた領域がガーベジとなる可能性がある。

例. (progn (setq x '(a b c)) (setq x (cdr x)))

TYPE 2:引数渡しによるもの

関数呼び出しにより引数が新たに束縛され、さらに前の引数の値が不要であるとき、前の値はガーベジとなる可能性がある。

```
例. (defun foo (x y)
      (cond ((null x) y)
            (t (foo (cdr x) (cons (car x) y)))))
    (foo '(1 2 3) '(4 5 6))
```

TYPE 3:ブロックからの脱出

progなどのブロックから抜けだすとき，porg変数のようなローカル変数に束縛されていた領域はガーベジとなる可能性がある。

例. (prog (x) (setq x '(1 2 3 4 5)) (f x))

TYPE 4:一時的領域確保

print関数のように領域が一時的に確保され、その関数の評価後すぐにその領域がガーベジとなる可能性がある。

例. (progn (print (list 1 2 3 4 5)) nil)

これらは参照カウンタ法で参照カウンタを減少させる操作がガーベジの生成する原因となるのと同様である。

３．ＧＣアルゴリズムの改良

我々の対象としたＧＣは一括型ＧＣである。一括型ＧＣには一般にフリーセルのリストを指し示しているポインタを持っておりこれがフリーセルの管理を行っている。即ち，ＧＣにより新たに１個セルを回収したときはフリーリストの先頭に加えることによりそのセルを使用可能にしている。

我々が行うＧＣの改良は以下のとおりである。まず、プログラムの静的解析を行うことにより、関数の評価後必ずガーベジとなることが知れる領域を見つけだし、その領域をフリーリストに加えるような働きを持つように関数自身を変換する。以下に例を示す。

例. (setq x y) -> (gc:setq x y)*1

もちろん、これは単純に変換してはいけない。なぜならば,xの値が共有されているかもしれないからである。

このように変換することによりガーベジの生成後、直ちにフリーリストに加えることができ、本来のＧＣの頻度が減少する。

以上の方法は文献(1)で採用されている方法と同様であるが、その対象とするＬｉｓｐプログラムの制限をなくし、対象を通常のCommon Lispとした。

４．静的解析及びその自動化

ここでは、ガーベジとなる領域をみつけるために前処理時に行うＬｉｓｐプログラムの静的解析について述べる。実際のプログラムに適用可能にするため、対象とする処理系のＬｉｓｐプログラムにはなるべく制限を設けない。なお、対象としたＬｉｓｐはCommonLispである。

機械による自動化を行うために機械化しやすいと思われる十分条件をいくつか得た。以下では、Ｌｉｓｐプログラムのリスト構造が木表現されているものとする。

４．１　単純解析及びその自動化

ガーベジとなる領域が比較的簡単な部分木のみで知れる場合がある。２.で述べたTYPE 4のガーベジ生成がそうである。この場合のガーベジとなる場所を機械的に見つけるには対象とする領域の位置から親を順に調べ、引数への束縛以外の束縛がなく,prog型の関数が見つかり、その位置が値を返す位置でなければ、その領域はガーベジとなる。これをまとめると，

*1 変換される関数はパッケージ"GC"にある関数と置き換えられる。"GC"にある関数は、ガーベジとなる領域をフリーリストに加え、本来の機能を実行する。
（これらの関数は効率を高めるためにsystem関数と同じように作成する。）

Garbage Collection by Static Analysis of Lisp Programs

Hiroshi Gomi[1], Hideki Nishigaki[1], Atsushi Nagasaka[2]
[1]Oki Techno Systems Laboratory. Inc.　[2]Oki Electric Industry Co..Ltd

(十分条件) 生成操作の親を順にたどっていき, prog型の関数にたどりつくまで、変数などに束縛されず、呼び出される関数に, special変数, global変数がなく, prog型の値を返す位置でなければよい。

prog型には, prog, progn, for, do, letなどがある。

4．2 必須順序項列

TYPE 1～3までのガーベジ生成を見つけるには共有の有無を調べる必要がある。領域の共有の有無を見つけるために、以下の項の列を定義する。即ち，与えられたＬｉｓｐプログラムに対して、項の列の先頭の項が評価されたなら必ずその項の列の順序で評価され、かつその項の列に他の項が入ることがない。またeval, applyなどの動的に環境が変化する関数の項も含めない。これを必須順序項列と呼ぶ。また、その中にあるユーザ定義関数を展開したのを展開された必須順序項列という。これらの詳細な定義は付録に示す。

(定理) 有限停止するプログラムにおいて展開された必須順序項列はsystem関数のみの項からなる。

この展開された項の列はsystem関数のみ、即ちその関数のふるまいがすべて知れているから、共有の判断は容易になり、また必ずこの順序で評価されるから、この項の列のみを考えに入れればよい。しかしながら、この項の列は制限が強いためあまり大きくなれない。そこで、ユーザ定義関数をlist的な関数に置換えることにより、共有は増加するが列の長さも長くなる。次の4.3で関数の引数と値の関係を調べることにより、共有を考えるだけにおいては、ユーザ定義関数を項の列の中から削除できる。

4．3 関数の値の解析

関数の値のデータタイプを解析することにより、引数と値が共有されない十分条件を見つけることができる。即ち、引数が構成型データタイプであり、値が要素型データタイプであるときは、引数と値は共有されない。これは文献(1)で取り入れられている方法である。 値のデータタイプを解析するために、関数本体を木表現し、部分木を展開する。そのさいに再帰的になった部分木は削除する。残ったの節はsystem関数のみになっているから、調べることができ、要素型、構成型、混合型、？*2のタイプに分類される。

4．4 ガーベジ条件

ガーベジ条件とはプログラム内の注目する位置において、ある時点で生成された領域が不要となるための十分条件である。以下にそのアルゴリズムを示す。

1. 生成操作を含む項を先頭とする必須順序項列を検出する。
2. ユーザ関数の引数の解析を行い、必須順序項列から削除できる項は削除する。
3. 項の列を展開し、system関数のみにする。
4. 項列の項を(setq var (f x_1 x_2 ⋯ x_n))の形の項列に変換する。但し, x_iは変数、または定数である。varは一時変数または通常の変数である。変換方法は明らかであるのでここでは省略する。
5. 一時変数も含めた変数が項列の各々の項において有効であるか否かを表す表（変数表）を作成する。
6. その表からガーベジの判断を行う。

5．システムの概要

本システムはユーザプログラムを前処理してガーベジとなる領域を見つけだし、ガーベジを処理する情報を付加したユーザプログラムに変換する前処理機械，ＧＣプログラムにガーベジの情報を渡しフリーリストに加えるようにする付加ＧＣ機械とパッケージ"gc"のなかの関数群から構成されている。

前処理機械は2パスで構成されており、pass-1でglobal変数, special変数の有無を調べ、ユーザ定義関数を項書き換え規則として表に登録し、生成操作も表に登録する。つづくpass-2では, pass-1で得られた生成操作の表をもとにして、その生成された領域がガーベジとなるか否かを以下の方法で調べる。まず、単純解析を行い、次に必須順序項列を検出し、展開し、ガーベジ条件を満たすか否かを調べる。

そして、ガーベジを生成する関数ｆをgc:fに変換する。gc:fはガーベジとなる領域のアドレスをＧＣのプログラムに渡し、フリーリストに加えて、本来のｆの機能を実行する。

6．おわりに

我々は各々のＬｉｓｐプログラムの特性に基づいたＧＣを行うようＧＣのアルゴリズムを改良した。本稿は文献(1)と同じ目的を持つが、本研究は一般のＬｉｓｐプログラムを対象にしたことが、異なっている。そのため、共有について複雑な計算を行う必要が生じた。そこで我々は必須順序項列を導入し、有効性が損なわれない程度に枝がりを行った。しかしながら、充分にガーベジを発見できたとは思えない。これが、今後の課題である。

最後に我々に研究の機会を与え、日頃御指導いただく総合システム研究所 椎野 努 部長に深く感謝致します。

参考文献

(1) 長谷川, 太田, 吉田, 福村, "LISPプログラムの記号的解析によるガーベージセル回収法とその実現", 信技報, 1984. EC83-52

(2) J. M. Barth, "Shifting Garbage Collection Overhead to Compile Time", CACM, 1977, vol. 20, no. 7

(3) H. Lieberman, C. Hewitt, "A Real-Time Garbage Collection Based on the Lifetime of Objects", CACM, 1983, vol. 26, no. 6

(4) Taiichi Yuasa, Masami Hagiya, "Kyoto Common Lisp Report", 1985, Kyoto University

付録 必須順序項列の定義

与えられた項ｔの任意の部分木ｓに対して、ｓを先頭とする必須順序項列ists(t, s)とは,

(1) s=変数, 定数のとき,
ists(t, s)=s, ists(t, next-position(t, s))

(2) s=f(t_1, t_2, ⋯, t_n), 但しｆの引数がすべて必須のsystem関数のとき,
ists(t, s)=ists(t, t_1), ists(t, t_2), ⋯, ists(t, t_n), s, ists(t, next-position(t, s))

(3) s=cond((p_1 f_1) ⋯ (p_n f_n))のとき,
ists(t, s)=ists(t, p_1)

(4) s=and(t_1, t_2, ⋯, t_n)のとき,
ists(t, s)=ists(t, t_1)

(5) s=f(t_1, t_2, ⋯, t_n), 但しｆは関数でユーザが定義したとき,
ists(t, s)=ists(t, t_1), ists(t, t_2), ⋯, ists(t, t_n), s

(6) s=f(t_1, t_2, ⋯, t_n), 但しｆはmacroでユーザが定義したとき,
ists(t, s)=s

system関数で他の特殊形式やmacroについては(3), (4)と同様にして求まる。next-position(t, s)はｔを探索していき, sの次に探索される部分木である。t_iは項である。右辺の外側にある","は列の連結を表す。

*2 evalのような動的に値のデータタイプが変化するものがあり、これらを？とする。それらも混合型と同じように扱う。