Bulletin of Aichi Univ. of Education, 62（Natural Sciences）, pp. 21 − 24, March, 2013

# Lisp言語処理系によるARCの評価

安本 太一

情報教育講座

# Evaluation of ARC on a Lisp System

Taichi YASUMOTO

*Department of Information Sciences, Aichi University of Education, Kariya 448-8542, Japan*

## 1 はじめに

ARC(Automatic Reference Counting)は，参照カウント方式のごみ集めの実装の一つで，2011年にApple社によって提案された［1, 2］.

ARCは，参照カウント方式のごみ集めのためのコード挿入を，コンパイル時に自動的に行うものである．一般に，参照カウント方式のごみ集めでは，データオブジェクトの保持や解放をカウンタに反映するためのコードをプログラマが挿入する必要があるが，これを自動的に行おうとするのがARCである.

参照カウント方式のごみ集めは，マークスイープ方式のごみ集めによって起こるプログラムの一時停止がないため，携帯電話やタブレット端末上のアプリケーションといったユーザと対話するリアルタイム性が要求されるアプリケーションに適しているとされている.

ただし，参照カウント方式の短所として，コーディングが面倒である事があげられる．ポインタの付け替えなどにおいて，データオブジェクトの保持や解放をカウンタに反映させるためのコードの挿入が煩雑で，これらのコードの挿入を誤ったり，忘れたりすることによる，バグが生じやすい.

このような状況を背景に，Apple社は，携帯電話やタブレット端末のためのiOSや，パーソナルコンピュータのためのOS Xのために，参照カウンタを操作するコードを自動挿入するARCを提案したと推測される.

過去の研究によれば，参照カウント方式のごみ集めとマークスイープ方式のごみ集めには，それぞれ，長所短所があり，どちらが良いとは断定されていない［3］．ARCが提供されるようになっても，OS X下では，マークスイープ方式のごみ集めも引き続き使用可能になっている.

本稿では，ARCをLisp言語処理系のプロトタイプに試験的に適用して，Lisp言語処理系によるARCの評価を行ったので，その結果を報告する.

## 2 ARCの概要

参照カウント方式のごみ集めは，あるデータオブジェクトが，他のいずれのデータオブジェクトからも参照されなくなったら，そのデータオブジェクトが占めていたメモリ領域を，再利用可能なメモリ領域として回収する.

カウンタが，データオブジェクト毎にあって，データオブジェクトへの参照の数（被参照数）を示している．新たに参照されるときはカウンタを増やし，参照されなくなるときはカウンタを減らす．OS XやiOSのための開発言語Objective-Cでは，カウンタの増減は次のように記述する.

```
[newval retain];
[currentval release];
currentval = newval;
```

currentvalにnewvalが参照している値を代入するときは，newvalをretainしてnewvalが指しているオブジェクトの参照カウンタを一つ増やし，currentvalをreleaseしてcurrentvalが参照していたオブジェクトの参照カウンタを一つ減らす．これまでの参照カウント方式のごみ集めでは，これらのコードをプログラマが挿入する必要があった.

ARCの場合は，これらのコードをコンパイル時に自動的に挿入してくれる.

```
//[newval retain];        ARCでは不要
//[currentval release];  ARCでは不要
currentval = newval;
```

さらに，currentvalが参照していたオブジェクトのインスタンス変数が，別のオブジェクトを参照していた場合は，ARCがその別のオブジェクトのカウンタを一つ減らすコードを自動挿入する.

オブジェクトの参照カウンタが0になったら，直ちに，そのオブジェクトが占めていたメモリ領域は再利

用可能なメモリ領域として回収される．

ARCを用いる場合でも，不要なオブジェクトが占めていたメモリ領域を直ちに回収するためには，積極的にコードに反映して，不要なオブジェクトを参照しない事をコンパイラにアピールしないといけない．例えば，currentvalが参照しているオブジェクトをもう使わないのであれば，次のようにNilをcurrentvalに代入するコードを積極的に書く必要がある．

```
currentval = Nil;
```

ある期間に生成されたオブジェクトが占有していたメモリ領域を一度に解放するために，@autoreleasepoolブロックというのがある．@autoreleasepool{ }の{と}の中で生成されたautoreleaseオブジェクトの占めていたメモリ領域は，このブロックの終了とともに解放され，再利用可能になる．先述のように，変数にNilを代入して，不要なオブジェクトを逐一解放するのではなく，一度に不要なオブジェクトを解放したい場合に適している．

## 3 処理系の変更

ARCを，Objective-Cで実装したLisp言語処理系のプロトタイプ［4, 5］に適用した．

このプロトタイプには，マークスイープ方式のごみ集めを使用している．OS Xではバージョン10.5以降から，マークスイープ方式のごみ集めが利用できるようになっている．OS Xのマークスイープ方式のごみ集めは，外部変数，静的変数，あるいはスタック内からたどることができる（C言語のデータでなく）Objective-Cのオブジェクトが占めていたメモリ領域は，回収しない．C言語の（動的に確保した）複合データやmallocで確保したような連続したメモリ領域は回収されてしまう．OS Xのマークスイープ方式のごみ集めを利用するためには，Lispのデータオブジェクトを表現する複合データは，Objective-Cのデータ型で実装するようにした上で，コンパイル時のオプションにマークスイープ方式のごみ集めを使用することを指定すればよいことになっている．これらのことに留意すれば，処理系を実装するプログラマは，マークスイープ方式のために，ごみ集めのためのコードを明示的に書く必要はない．

このLisp言語処理系のプロトタイプにおいて，ARCを使用する場合は，コンパイラに与えるオプションをARCを使用するように変更するだけでよいはずである．ARCによるメモリ管理の対象も，マークスイープ方式と同様に，Objective-Cのデータである．そして，このLisp言語処理系のプロトタイプのLispのデータオブジェクトは，既に，Objective-Cのデータで表現されているからである．

ARCのためにわざわざ行うべきことがあるとすれば，演算などが終わりLispのデータオブジェクトが必要なくなるような箇所においては，すみやかに，そのオブジェクトを参照している変数にNilを代入することである．

しかしながら，実際には次のような問題が生じた．ARCを使用するようにコンパイラへのオプションを指定すると，Lisp言語処理系のプロトタイプのコンパイル時に，セレクタ（メッセージセレクタ，あるいはメソッド名ともいう）についての多くのエラーがでた．Lispの各データ型オブジェクトのインスタンスを，（各データ型のオブジェクトのインスタンスを指し示す変数ではなく）Lispの各データ型を総称する上位クラスのインスタンスを指し示す変数で参照して，メッセージ送信を行っている箇所で，“明示的なセレクタがない”というようなエラーになる．マークスイープ方式でコンパイルした時は，“そのようなセレクタはないかもしれない”という警告にとどまっていた．

Lisp言語処理系においては，メンバやメソッドがほとんど定義されない上位クラスのオブジェクトで，（この上位クラスを継承して定義したサブクラスの）Lispの各データ型オブジェクトを総称することはよく行われるので，このようなエラーには困惑した．これは，上位クラスのオブジェクトに，形式的なセレクタを定義することによって，とりあえず解決した．

また，同じく，上位クラスでLispの各データ型のオブジェクトを総称することにかかわって，代入をしているところで，ポインタの型が一致しないといった警告も多数出るようになった．

## 4 評価実験

ARCの性能を，マークスイープ方式のごみ集めと比較した．

### 4.1 実験方法

Objective-Cで実装したLisp言語処理系のプロトタイプにARCを適用した．テスト用のプログラムを実行し，その実行時間，実メモリ，スレッド数を計測した．スレッド数は，Objective-Cコンパイラ及びランタイム環境に任せた結果であり，筆者が指定したものではない．

ARCの性能を評価するために，Apple社製コンピュータMac上で，下記のLispの式を10000000回繰り返して評価するテストプログラムの実行時間を計測した．

```
(+ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
```

今回の実験は，ごみ集めの性能評価であることから，Lispの式を構成するコンスセルと整数のデータオ

ブジェクトは，繰り返しのたびに生成と破棄を繰り返すように，処理系を変更した．その他，処理系には，最小限の組み込み関数だけを登録するなど，処理系そのものが動作するために最低限必要なメモリが少なくなるようにし，コンスセルやデータオブジェクトのためのメモリの消費が支配的になるように努めた．実行コードの生成に用いたコンパイラはXcode 4.41のLLVMのclangで，最適化オプションは`-Os`を指定した．

評価実験に用いたMacは，表1に示す3台であり，CPUやメモリ伝送速度などが異なっている．メモリ伝送速度は，Mac mini Server 2010が最も遅い．筆者が用意したメモリを読み書きするテストプログラムを用いた予備的な実験によると，Mac mini Server 2010の実質的なメモリ伝送速度を基準として1とすると，Mac Pro Early 2009は1.16倍，Mac mini Sever 2011は1.57倍速かった．各Macの正式名称は，この段落や表1で用いているものであるが，以後，混乱を招かない範囲で省略名称を用いる．

表1：評価実験を行った実行環境

| 実行環境 | CPU | メモリ | キャッシュ |
|---|---|---|---|
| Mac Pro Early 2009 | 2.26GHz Xeon5500x2 8コア | 12GB 1066MHz DDR3 | L2:256KB L3:8MB |
| Mac mini Server 2010 | 2.66GHz Core 2 Duo 2コア | 8GB 1067MHz DDR3 | L2:3MB |
| Mac mini Server 2011 | 2GHz Core i7 4コア | 8GB 1333MHz DDR3 | L2:256KB L3:6MB |

（キャッシュのL2は2次，L3は3次）

### 4.2 実験結果

ごみ集めとしてARCを用いた処理系とマークスイープ方式を用いた処理系を用意し，上記のテストプログラムを実行したときのそれぞれの実行時間を表2に示す．

表2：マークスイープ方式とARC方式の実行時間の比較

| 実行環境 | マークスイープ | ARC | 比 |
|---|---|---|---|
| Mac Pro 2009 | 4:21 | 4:38 | 1.07 |
| Mac mini 2010 | 4:23 | 5:27 | 1.25 |
| Mac mini 2011 | 4:01 | 4:31 | 1.12 |

（実行時間の単位は分:秒）

いずれの実行環境においても，ARCの方が実行時間が長く，7%から25%実行時間が増加する．実行時間が最も増加しているのは，メモリ伝送速度が最も遅いMac mini 2010である．

また，評価プログラムを実行したときの実メモリとスレッド数を表3に示す．実メモリとスレッド数の計測は，OS X Mountain Lionに付属のユーティリティであるアクティビティモニタを用いた．アクティビティモニタはGUIアプリケーションで，評価プログラムの実行開始後，実メモリやスレッド数が落ち着いたと筆者が判断したときの数値を記録した．計測のタイミングを厳密に定めていないので，傾向を知るための大まかな値である．

表3：マークスイープ方式とARC方式の実メモリとスレッド数

| 実行環境 | マークスイープ | | ARC | |
|---|---|---|---|---|
| Mac Pro 2009 | 14.3MB | 12 threads | 1.9MB | 1 thread |
| Mac mini 2010 | 11.5MB | 3 threads | 1.7MB | 1 thread |
| Mac mini 2011 | 14.3MB | 7 threads | 1.8MB | 1 thread |

3つの実行環境で共通しているのは，ARCは，マークスイープ方式に比べて，メモリの使用量が7分の1程度と圧倒的に少ないことと，スレッド数が1であることである．

マークスイープ方式の場合はマルチスレッドで実行されるが，コア数が多いCPUほど，多くのスレッドで実行されているようである．マルチスレッドによる実行については，コンパイラによる最適化あるいはObjective-Cやマークスイープ方式のごみ集めのランタイム環境のサポートによるものだと推測される．筆者は，Lisp言語処理系のプロトタイプのコーディングにおいて，明示的にマルチスレッドを利用していない．

## 5 考察

評価実験においては，ARCの方がマークスイープ方式のごみ集めより，実行時間が増加する結果となったが，オブジェクトの参照カウンタ操作が原因だと思われる．メモリ伝送速度が遅いコンピュータにおいては，この参照カウンタの操作が重荷になっていることがうかがえる．

評価実験が行われた状況においては，引数となっている個々の整数オブジェクトは演算が終わり用済みになると，式全体の計算が終わっていなくても，その整数オブジェクトの参照カウンタを直ちに減らすようになっている．Lispの式を構成しているコンスセルオブジェクトについては，式全体の計算が終わってから，`@autoreleasepool`ブロックの機能を用いてまとめて解放するようになっている．`@autoreleasepool`ブロックの機能を使わずに，不要となったコンスセルの参照カウンタを逐一減らすようにしていたら，実行時間がさらに増える事が推測される．

実行時間の増加の原因として，もう一つ挙げられるのは，シングルスレッドで実行されることである．参照カウンタの操作は，カウンタ増減の順序が大切であるので，並列実行に向かず，コンパイラによる最適化やランタイム環境のサポートは期待できないと推測される．逆の見方をすると，ARCはシングルスレッドで実行されているわりには，健闘しており，マルチスレッドで実行されているマークスイープ方式に比べて

2倍も3倍も遅いわけではない．

使用する実メモリは，ARCの方が，マークスイープ方式のごみ集めより，圧倒的に少ない．ARCの基本となっている参照カウント方式の込み集めの典型的な特徴であろう．評価実験では，実行時間が最大25%増しになることがあったが，使用する実メモリが約7分の1になっていることは十分評価に値する．

## 6 まとめ

Lisp言語処理系のプロトタイプにおいて，ARCとマークスイープ方式のごみ集めをそれぞれ適用して，ARCの性能評価を行った．

ARCの長所は，手軽に参照カウント方式のごみ集めを導入でき，使用する実メモリを，マークスイープ方式のごみ集めより，大幅に減らす事ができることである．短所としては，マークスイープ方式より実行時間が増加する可能性があることである．マークスイープ方式のように，スイープ時に全メモリを走査することがないので，ARCの方が実行時間が短くなる可能性を否定するわけではないが，本稿で行った評価実験のように実行時間が長くなるケースがあるのも事実である．

ARCは，参照カウンタの操作を行うコードを自動挿入する機能ではあるが，万能ではない．先述のように，ARCを使用する事を選択してLisp言語処理系をコンパイルすると，マークスイープ方式では寛容されていたコードが，エラーや警告になったりすることがある．メンバやメソッドが実際に存在するか否かを（コンパイル時に判断するのではなく）実行時に判断する動的な側面で，このようなエラーや警告がでることがある．

厳密なコードを書くことは大切であるが，参照カウンタのためのコードが自動挿入になっても，些細なことでエラーや警告になるのは，Objective-Cの動的な側面を期待しているプログラマには負担であろう．その他，今回，ARCを使用してLisp言語処理系を実装していく過程において，評価プログラムを実行しているときメモリリークがおき，使用している実メモリが9GBにも達することがあった．原因がわからず，`@autoreleasepool`ブロックを用いて，対処した．

ARCを適用したLisp言語処理系の使用経験をさらに積み，少ないメモリで動作するLisp言語処理系の性能向上に努めることが今後の課題である．

## 参考文献

[1] *Transitioning to ARC Release Notes*, Apple（2011）.
[2] 荻原剛志：詳解Objective-C 2.0第3版，ソフトバンククリエイティブ（2011）.
[3] 中村成洋，相川光：ガベージコレクションのアルゴリズムと実装，秀和システム（2010）.
[4] 安本太一：Objective-CによるLisp言語処理系の試験的実装，愛知教育大学研究報告自然科学編，五十八輯，pp. 21–25（2009）.
[5] 安本太一：Objective-CによるLisp言語処理系のメソッド呼出しの高速化，愛知教育大学研究報告自然科学編，五十九輯，pp. 19–22（2010）.

（2012年9月18日受理）