Common ESP 処理系におけるガーベジコレクタの実現

5J-5

田中吉廣 実近憲昭 田中康之

(株)AI言語研究所 (財)日本情報処理開発協会

1 はじめに

Common ESP(*CESP*)は論理型言語とオブジェクト指向を融合した言語であり、AIプログラムの開発、プロトタイピングを含むシステム記述に適している。CESPにおける記述能力の高さはデータを自由にまた動的に生成できる点によるところが大きい。一方、このようなデータを扱う言語のメモリ管理機構には、実行の経過により不必要となったデータ領域を回収するガーベジコレクタ(塵集め機構:*GC*)が不可欠である。本稿では平成元年度に開発された基本仕様版CESPに実装したGCについて紹介する。

2 データ構造とその管理方式

CESPではPrologの持つデータ構造のほかに本言語の各種機能を実現するために様々なデータ構造[4]が導入されている。アトミックなデータとしては、アトム、整数、実数(浮動小数点数)があり、変数は未束縛変数の他に遅延実行制御変数(**Hook**)がある。構造体には、副作用がないものとしてリスト、ベクタ、項、ピュア文字列があり、副作用があるものとしてヒープベクタ、文字列、凍結構造(**Frozen Data**)、オブジェクト、ハッシュテーブル、他言語I/F構造(**Foreign**)がある。

副作用のあるデータのうち、他言語I/F構造以外はヒープ領域上にのみ配置される。他言語I/F構造はヒープ/グローバルスタック以外の領域に配置される。これに対し、副作用のないデータは一般にグローバルスタック上に配置されるが、変数以外のデータは、ヒープ領域に置かれることもある。両領域に展開されたデータ構造は一般的なPrologと同様に引数レジスタ、トレイルスタック、選択点スタック、環境スタックより指し示されるが、CESPにおいては更にヒープポインタテーブル、間接語テーブルからもヒープ領域を指している。

さらにCESPではGC処理の判定用のビットをデータ中に保持できないために、GCテーブル領域を持っている。このテーブルは、各領域の1ワード毎に3ビットが対応し、その各々は、塵でないデータであることを示すマークビット(*m-bit*)、ポインタでないデータを示す基底ビット(*b-bit*)、ポインタデータの再配置を行う際に使用する逆転ビット(*f-bit*)と呼んでいる。

3 GCの実現方式

3.1 GCの基本アルゴリズム

今回CESPに対し導入したGCは一括型GCの一つであるマーク&コンパクション方式によるGC[1]である。CESPでは先に述べたようにデータ構造を保持する領域としてグローバルスタックとヒープ領域があるが各々の領域に対し必要なサイズのデータ領域が確保できない時にGCが起動される。

マーク処理では現在有効なデータ構造すべてにマークを付ける。この時ルートとなるものは、グローバルスタックのGCでは引数レジスタ、トレイルスタック、選択点スタック、環境スタックであり、ヒープ領域のGCでは更にヒープポインタテーブルと間接語テーブルもルートとしてマーク処理を行う。Prologではトレイルスタックはマーク処理のルートとしなくて良いが、CESPではトレイルスタック上に遅延実行制御データやバックトラック時にゴールを実行するための制御構造が保持されているため、これをルートとしてマークを行う必要がある。

コンパクション処理は更にスィープ処理とコンパクト処理に分けられる。スィープ処理ではGCの対象領域を参照する外部からのポインタや対象領域内で参照し合うポインタを更新するための処理を行う。具体的には更新を必要とするポインタの参照方向を逆転しておき、後のコンパクト処理の段階で該当セルの移動場所が確定した際にそれらのポインタ内容を更新する。コンパクト処理ではマーク処理の際検出された不要領域の除去を行う。この処理は2パスで行われ、最初のパスで外部からの参照とGC対象領域内の後ろ向きの参照を解決し、次のパスで不要セルの回収および領域内での前方参照を解決する。

3.2 最適化処理

CESPに実装したGCの基本アルゴリズムは先に記述した通りであるが、本年度は以下の最適化を導入した。

3.2.1 一時変数の早期回収

次のようなプログラムについて考えてみる。

```
program(B) :- p(A,B);
p(A,B) :- create_structure(A),q(B);
p(A,B) :- some(A),r(B);
```

このプログラムに対してprogram(B)が呼ばれると、pの2番目の節の呼出しのために選択点が生成される。しかし、このAが現在有効な環境の連鎖から参照されていない、例えばq(B)が呼び出された後はもはや不要なものである。この

Implementation of Garbage Collector on Common ESP

Yosihiro TANAKA[1], Noriaki SANECHIKA[1], Yasuyuki TANAKA[2]

1: AI Language Research Institute,Ltd. 2: Japan Information Processing Development Center.

ような領域を回収する手法については[1]に詳しい。

3.2.2 コンパクション対象領域の削減

この手法はGC処理が2回以上行われた場合、対象領域の上位部分には回収できない(塵ではない)領域が連続するという期待に基づいている。なぜならば前回までのGCによって保存しておく必要があるセルはコンパクション処理によって上位部分に集められるからである。

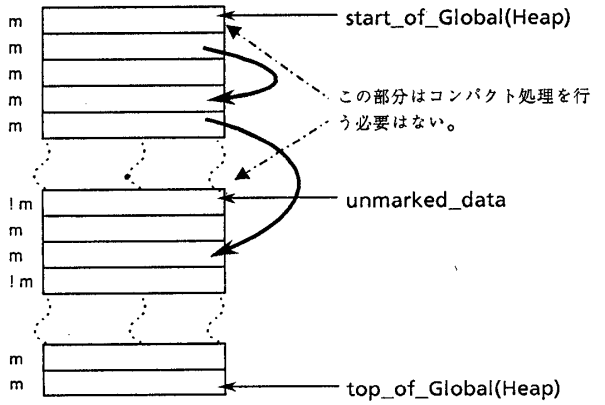

図1: コンパクション不要領域

マーク処理後の記憶領域を上位から見て最初のマークされていないセルのアドレスをunmarked_dataとすると、start_of_Global(Heap)からunmarked_dataまではコンパクト処理後も移動することはなく、実際にコンパクト処理の対象としなければならないのは、unmarked_dataからtop_of_Global(Heap)の間であることが分かる。また、前者は後者を参照するポインタのみを更新するためにスィープ処理だけを行えばよい。

4 性能評価

性能評価はSUN3-260上で行った。GC自体はC言語で記述されている。ヒープ/グローバルのサイズはともに100Kワード(400Kバイト)として、あるプログラムのコンパイル中に発生したGCの処理時間と回収量を記録した。

表1: グローバルスタックの測定結果

		基本GC	一時変数		領域削減		総合GC	
			計測値	改善率%	計測値	改善率%	計測値	改善率%
1	時間(秒)	7.60	5.93	128.1	8.37	90.8	6.20	122.6
	回収率%	80.63	89.21	110.6				
2	時間(秒)	7.88	6.33	124.5	6.02	131.0	5.77	136.7
	回収率%	77.08	84.85	110.1				
3	時間(秒)	8.55	7.30	117.1	6.13	139.4	6.68	127.9
	回収率%	72.94	79.58	109.1				

計測結果をみると一時変数の早期回収による最適化はグローバル領域のGCにのみ効果があったことを示している。これは今回の測定条件の下での特有のものであり、グローバル領域からの参照が一時変数の早期回収により解除されればヒープ領域も回収される。しかしながら、一時変数の早期回収を取り入れたGCと基本アルゴリズムの比較は容易には行えない。なぜならば、GC処理は領域が不足したときに起動されるため回収率が異なると起動される時期も異なるからである。一時変数の早期回収による最適化については1回目だけを基本アルゴリズムと比較するべきであろう。

表2: ヒープ領域の測定結果

		基本GC	一時変数		領域削減		総合GC	
			計測値	改善率%	計測値	改善率%	計測値	改善率%
1	時間(秒)	12.97	12.55	128.1	12.45	104.2	12.62	102.8
	回収率%	3.30	3.30	100.0				
2	時間(秒)	13.93	13.40	104.0	5.27	264.6	5.33	261.3
	回収率%	2.50	2.50	100.0				
3	時間(秒)	14.20	14.72	96.5	4.98	285.0	5.88	241.4
	回収率%	2.79	2.79	100.0				

コンパクション対象領域の削減による最適化は特にヒープ領域のGCにおいて効果があった。これはヒープ上にはCESP起動時の初期データやコンパイルされたオブジェクトコードに含まれる定数データ、スロットに設定されたデータなど計算の進行過程と関連の少ないデータが数多く存在し、GCが行われた後はヒープの上位に集中するためと考えられる。また、マークされていないセルを検出するのにかかる費用はコンパクト処理を行うのに比べて格段に少ないため計算の過程により大きく内容の異なるグローバルスタックのGCにおいても若干の改善が見られることが分かった。

5 まとめ

以上、基本仕様版CESPにおけるGCの実現について述べた。今回導入した方式は通常の実行には負担がなく不要なデータは全て回収できるといった利点を持つが、反面、一旦回収が始まると長時間にわたりプログラムの実行が中断されるといった欠点をもっている。この欠点に対して不要となった単一参照を実時間で回収するMRB (Multiple Reference Bit)方式のGCを併用するなどは将来の課題である。また、一括型GCの実行性能の改善のため、計算の進行過程に依存して領域をセグメントに分割する世代管理方式を近く導入していく予定である。

6 参考文献

[1] Appleby,K. Carlsson,M. Haridi,S. and Sahlin,D. : Garbage Collection for Prolog Based on WAM, Communication of the ACM 31,6 (Jun 1988),719-741

[2] Jonkers,H.B.M. : A Fast Garbage Compaction Algorithm. Information Processing Letters 9,1 (Jul 1979) 26-30

[3] Morris,F.L. : A Time- and Space- Efficient Garbage Compaction Algorithm, Communication of the ACM 21,8 (Aug 1978),662-665

[4] 清水ほか "Common ESP処理系の内部データ形式の実現" 情報処理学会第39回全国大会(1989)