Variable Length Storage Allocator とその Garbage Collector*

二村良彦**　桜木邦子**

要　旨

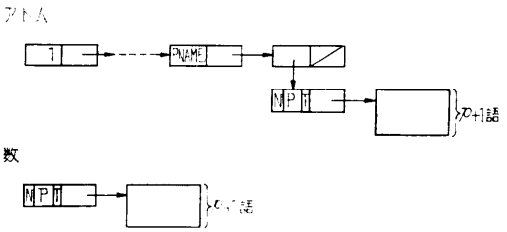
本文は可変長の記憶場所を割付けるストレッジ・アロケータで用いる記憶場所の管理方法，および使い捨てられた記憶場所を再生利用する方法について述べるものである．このストレッジ・アロケータは LISP 1.5 と同じように，記憶領域（storage area）を free storage と full word space の二つに分けて各々を異なった方法で管理する．特に full word space においては，使い捨てられた記憶場所を集める際に，記憶場所の詰め替えをする．本文で述べる方法は，LISP のように，使い捨てられた記憶場所を自動的に再生利用する機構を持つ記号処理システムを，bit 演算命令を備えた計算機において作成する際に有効である．

1. 緒　言

リスト構造における multi-word item の使用による利益は，リスト処理に要する時間，およびリスト構造のための記憶場所（storage）の節約である．われわれは，HITAC 5020 において LISP 1.5[2] と類似の記号処理システム HELP (Hitachi Experimental List Processor) を作成する際に，BCD character string や数値をしまうリスト要素は，多重語からなるブロックとすることにした．そのためストレッジ・アロケータは，可変長の記憶場所を割付けたり，使い捨てられた多重語からなるブロックを再生利用しなければならなかった．本文は，そこで用いられた記憶場所割付けの方法に関して述べるものである．

HELP におけるアトムおよび数は，**第 1 図**に示す構造を持つ．

この構造は，BCD character string や数のように，リストに分解するよりまとめてブロックにしておいた方が処理しやすいデータは，ブロックとして記憶するという考えに基づく．HELP ストレッジ・アロケータはこのようなリスト構造を作るために，可変長の記憶場所を割付ける．それは，文献 2) におけると同じく記憶領域を，pointer word（ポインタを含む語）のための free storage と，full word (BCD charactor string や数を含むブロックのための語）のための full word space の二つに分けて管理する．システム実行中に記憶領域が尽されると，HELP garbage callector を自動的に呼んで，使い捨てられた記憶場所を集めて再び使えるようにする（この操作を“ゴミ集め”と呼ぶ）．garbage collector は free storage が尽された時は free storage の中だけでゴミ集めを行ない（リスト構造を作るために用いられる pointer wordの数が増えても，そこで用いられる full word の数が増えるとは限らないので，この場合に full word space のゴミ集めをする必要はない），full word space が尽された時は free storage と full word space の両方にわたってゴミ集めを行なう．full word space のゴミ集めの際には，隙間をうめるためにブロックの詰め替えをする．



第 1 図　HELP におけるアトムと数の構造；

N は 5 bit-field, P は 8 bit-field, そして T は 3 bit-field である（HITAC 5020 の 1 語は 32 bit 構成）．Nの内容はその半語（16 bit-field）がポインタでないことを示す特定の数値，P の内容は，語の右半語で指されるブロックの大きさを示す数値（図ではPの内容は p, $0 \leqq p \leqq 255$）．そしてTの内容は，数のタイプを示す数値である．

2. 記憶領域（storage area）の管理

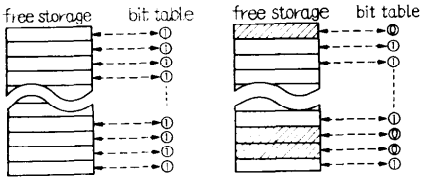
* Variable Length Storage Allocator and its Garbage Callector, by Yoshihiko Futamura and Kuniko Sakuragi (Hitachi Central Research Laboratory)

** 日立中央研究所

HELP ストレッジ・アロケータは，記憶領域を pointer word（ポインタを含む語）のための free storage と，full word（BCD charactor string や数を含むブロックのための語）のための full word space の二つに分けて，各々を別の方法で管理する．

2.1. Free storage の管理

free storage に対して bit table をとり，free storage の各語と bit table の各 bit との間に1対1の対応をつける．語が空いているときは対応する bit を 1，語が使われると対応する bit を0にする（**第2図**参照）．



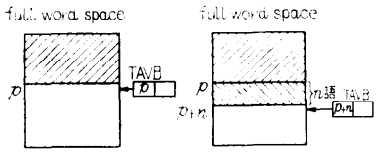
第2図 free storage と bit table の対応

free storage の全ての語があいているとき，bit table の全ての bit は 1（左側），語が使われると対応する bit は0になる（右側）．図で斜線の引かれた語は使われている語を示す．
bit table は記憶領域と異なる（システム内の）場所にとられる．

HELP ストレッジ・アロケータは，pointer word が必要になると bit table における一番始めの 1 の bit に対応する語をとってきて，しかるのちその bit を0にする（HITAC 5020 には，32 bit または 64 bit field における一番左の 1 に対応する bit をさがし出し，しかるのちその bit を0にする機械語の命令（FTL）があるので，この方法は free storage をリストにして管理する普通の方法より有効である）．bit table の全ての bit が0のときは，garbage collector を呼ぶ．

2.2. Full word space の管理

full word space に対しては bit table をとらず，システム・レジスタ TAVB（Top of AVailable Block）によって一番始めの空き語を指す．HELP ストレッジ・アロケータは，BCD charactor string や数をしまうための n 語ブロックが必要になると，レジスタ TAVB によって示される語から n 語とってきて，しかるのち TAVB の内容を n 増やす（**第3図**参照）．full word space に必要なだけの空き語がないと，garbage collector を呼ぶ．



第3図 full word space とTAVB

TAVB は full word space における一番初めの空き語を指す．左図の状態で n 語使われると右図の状態に移る．

3. HELP garbage collector

〔定義〕

使われている語（またはブロック）：システム・レジスタ BASE 1, BASE 2, …, BASE l の左半語からポインタによってたどれる記憶領域内の語（またはブロック）

使い捨てられた語（またはブロック）：使われている語以外の記憶領域内の語（またはブロック）

3.1. Free storage が尽された時のゴミ集め

（1） bit table の全ての bit を1に，i を1に，そしてスタックを空に設定し，（2）を行なう．

（2） $i>l$ ならば，ゴミ集めをやめる．

$i \leq l$ ならば，BASE i の左半語のポインタ（左ポインタ）で指される語を調べるべき語として（3）を行なう．

（3） 調べるべき語を x とする．x が pointer word のとき，x に対応する bit が 0 ならば（4）を行ない，x に対応する bit が1ならばその bit を0にし，かつ x を調べるべき語として（5）を行なう．

x が pointer word でなければ（4）を行なう．

（4） スタックが空ならば，i を $i+1$ で置き換えて（2）を行なう．

スタックが空でなければ，スタックの一番上の語の右ポインタ（右半語の内容）で指される語を調べるべき語とし，かつスタックを pop up して（3）を行なう．

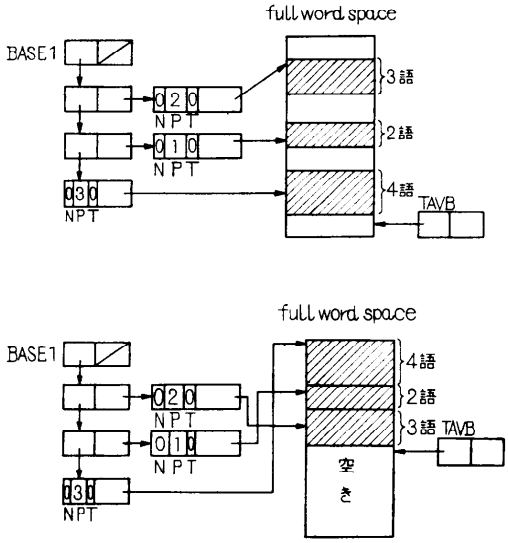
（5） 調べるべき語を y とする．y をスタックし，かつ y の左ポインタで指される語を調べるべき語として（3）を行なう．

以上の（1）～（5）により free storage の全てのゴミ（使い捨てられた語）に対応する bit は1に，使われている語に対応する bit は0に設定される．

3.2. Full word space が尽された時のゴミ集め

full word space が尽されると garbage collector は，使われているブロックだけを full word space

の始めから隙間が空かないように詰めかえる．詰めかえが終ると，システム・レジスタ TAVB を空き語の先頭を指すように設定する（**第4図**参照）．



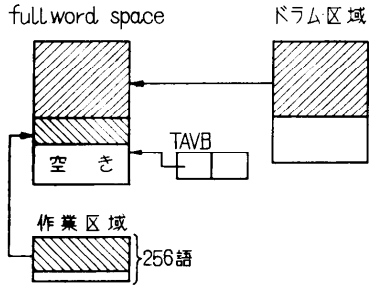
第4図　full word space のゴミ集めの前後の状態

上の状態でゴミ集めが起ると下の状態にかわる．（ただし，この図では $l=1$ とした）

使われているブロックの詰めかえの際に，作業区域（コア 256 語）とドラム区域（full word space より 256 語少なくドラム内にとる）を用いる．

（1） bit table の全ての bit を1に，i を1に，スタックを空に，システム・レジスタ TAVB の左ポインタを full word space の先頭番地に，そして作業区域とドラム区域を空に設定して（2）を行なう．

（2） $i>l$ のとき，ドラム区域で使われている場所の内容を full word space の先頭からロードし，そのすぐ次の番地以降に作業区域で使われている場所の内容を移す（**第5図**参照）．



第5図　ドラム区域および作業区域の斜線を引かれた部分（使われている場所）が full word space に矢印で示すように移る．

しかるのち，TAVB の左ポインタを一番始めの空き語に設定してゴミ集めをやめる．

$i \leq l$ ならば，BASE i の左ポインタで指される語を調べるべき語として（3）を行なう．

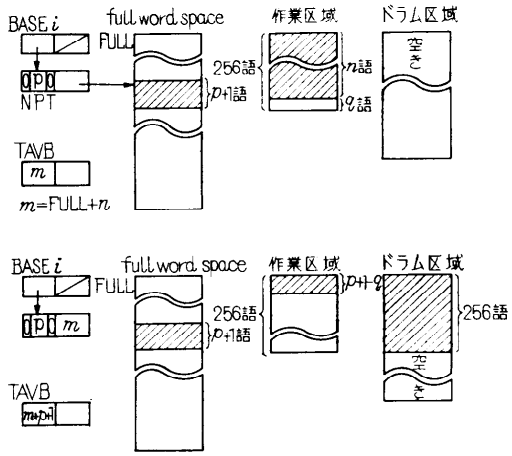
（3） 調べるべき語を x とする．x が pointer word のとき，x に対応する bit が0ならば（4）を行ない，x に対応する bit が1ならばその bit を0にし，かつ x を調べるべき語として（5）を行なう．

x が pointer word でなければ，（4）を行なう．

（4） スタックが空ならば，i を $i+1$ で置換えて（2）を行なう．

スタックが空でなければ，スタックの一番上の語の右ポインタで指される語を調べるべき語とし，かつスタックを pop up して（3）を行なう．

（5） 調べるべき語を y とする．y の右ポインタが full word を指していれば，その full word から始まる y の P-field の内容より1大きい語数のブロックを，作業区域の空き場所の先頭から詰めこむ．y の右ポインタを TAVB の左ポインタで置きかえ，しかるのち TAVB の左ポインタをブロックの大きさだけ増やし（4）を行なう．作業区域が一杯になったら，その内容をドラム区域の空き場所に移し，次の詰めこみを作業区域の先頭から行なう（**第6図**参照）．



第6図　作業区域へのブロックの詰込

上図で $p \geq q$ のとき詰めこみが起こると，ブロックの q 語を作業区域に詰め，次に作業区域の内容をドラム区域に移し，最後にブロックの残りの $p+1-q$ 語を作業区域の先頭から詰める（下図）．

yの右ポインタが full word を指していなければ，yをスタックし，かつyの左ポインタで指される語を調べるべき語として（3）を行なう．

以上の（1）～（5）により，使われている語に対応する全ての bit は0に，使い捨てられた語に対応する全ての bit は1に設定され，かつ使われているブロックだけが full word space に詰められる．

4. 結　　言

本ストレッジ・アロケータは LISP と類似の記号処理システム HELP (Hitachi Experimental List Processor) で用いられているものである．full word space のゴミ集めの際にドラムを用いるので，free storage 9K 語，full word space 2K 語のゴミ集めに HITAC 5020（コア・アクセスタイム 2 μs，ドラム・アクセスタイム 10 ms）で約 1.5 秒かかる．

しかし，非数値的な計算の実行によって費やされる記憶場所のほとんど全ては pointer word であるから，普通の記号処理問題において full word space のゴミ集めが起こることはまれである．したがって，そのゴミ集めに少しぐらい時間が多くかかっても，次の点を考慮すると，本文で述べた方法は有効である．（1）BCD charactor string や数の処理時間（入出力ルーチンや算術ルーチンにおける），およびそれらのための記憶場所の節約．（2）free storage のゴミ集め時間の節約．

終りに臨み，システム作成の際に適切な助言をして下さった日立中研の嶋田正三氏と吉村一馬氏に厚く御礼申し上げたい．

参考文献

1) W.T. Comfort: Multiword List Items, CACM, Vol. 7, No. 6, Jun, (1964)
2) J. McCarthy 他: LISP 1.5 Programmer's Manual, MIT Press, Cambridge Mass., (1962)

（昭和42年3月15日受付）