14. リストのコピー法†

長谷川　洋††

1. はじめに

リスト構造 (list structure) に何んらかの操作をする場合，一般には処理対象であるリスト構造の他に，**スタック** (stack) や**キュー** (queue) などを使用してその処理を行っている．そこで用いられるスタックやキューなどは，処理を遂行する上で補助的に用いられる作業用の記憶領域であるため，**補助作業領域**と呼ばれる．

補助作業領域を用いて**リスト処理**を行うと，ほとんどの場合に，容易に高速な処理が実現できる．しかしその一方で，ほとんどのリスト処理アルゴリズムは補助作業領域を使用せずとも実現可能であると言われており[17]，また，

（1） 多大の補助作業領域を使用することが重大な欠点となる場合や，

（2） 補助作業領域を使用しないアルゴリズムの方が，それを使用する場合よりも高速な処理を実現している場合

がある．例えば(1)の例として，記憶領域がなくなった時に不要となっている記憶領域を回収するために行う**回収再生法** (garbage collection) では，その第1フェーズである**リストの印づけ** (list marking) において，印づけアルゴリズム自身が多大の補助作業領域を使用することは絶対に回避しなくてはならない．(2)の例としては，**LISP**[14] の read 関数，リストの反転 (reverse) や**リストの移動**[2] などが挙げられる．これらのアルゴリズムは，補助作業領域を使用する同一機能のアルゴリズムと比較して，アルゴリズム自身の占める記憶容量の大きさはほとんど等しく，補助作業領域を使用しないためにアルゴリズム自身が巨大化しているということはない．

一般に，プログラム変数を除いては，補助作業領域を一切使用しないリスト処理アルゴリズムを**固定作業領域** (constant または bounded workspace) **リスト処理アルゴリズム**と呼び，活発な研究が行われている．本稿で述べる**リストのコピー**についてもそのすべてが固定作業領域アルゴリズムであり，そして，そのほとんどが，補助作業領域を使用する場合よりも高速な処理を実現している．

次節では，リスト構造の分類と，コピーが作成される領域の分類を行う．3節では，代表的な固定作業領域リストコピーアルゴリズムである Robson[15] のアルゴリズムについて解説し，種々のリストコピーアルゴリズムについては，4節で概観する．

2. リスト構造

2.1 リスト構造

リスト構造は，car, cdr と呼ぶ2つの**ポインタ**を含む**セル**から構成され，car, cdr ポインタは，それぞれ任意のリストまたは**アトム**と呼ぶ非リストを指す[14]ものとする．car, cdr ポインタでリストを指すものをリストポインタ，アトムを指すものをアトムポインタと呼ぶ．コピーされるリスト構造を原リスト (orignal list) と呼び，コピーされたリスト構造のことをコピーリストまたはコピーと呼ぶ．同様に，原リストを構成するセルを原セル，コピーリストを構成するセルをコピーセルと呼ぶ．原セルとコピーセルのそれぞれの car, cdr に入る最終値を，oldcar, oldcdr, copycar, copycdr と呼ぶ．アトム自身はコピーされないので，それゆえ，アトムポインタの oldcar と copycar，または oldcdr と copycdr とは等しく，同一のアトムを指す．また原リストは**根** (root) により指されている．

任意のリスト構造は，**木** (tree)，**共有部分木を持つ木** (tree with shared subtree)，**再入リスト** (reentrant list)，**リスト**に分類される[12]．いま，**図-1** に示すリストを root, car, cdr と**前順序** (preorder) で重

† Algorithms for Copying List Structures by Hirohi HASEGAWA (Programming Research Section, Computer Science Division, Electrotechnical Laboratory).

†† 電子技術総合研究所ソフトウェア部プログラム研究室

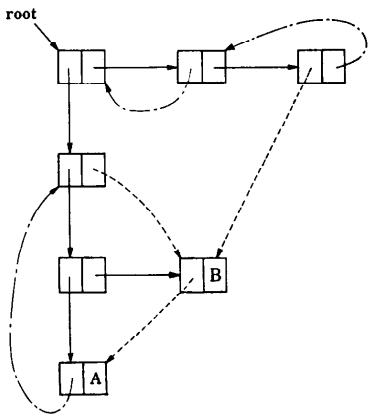

図-1 リスト
(実線はリストポインタ，破線は共有ポインタ，
一点鎖線は再入ポインタを示す.)

複なくたどれば，それにより，リスト構造の骨組を成す**極大木** (spanning tree) が決定される．ここでは，極大木を構成するリストポインタをFポインタと呼び，図-1 では実線で示す．ある部分木 (subtree) または部分リスト (sublist) が2つ以上のポインタによって指されている時，その部分木または部分リストは**共有** (share) されていると言い，それらのポインタでFポインタでないものを**共有ポインタ**と言う．また，あるセルに含まれるポインタが，そのセルを含めて，根からそのセルまでの道筋にある任意のセルを指す時，そのポインタを**再入ポインタ**と言う．そこで，木は極大木を構成するFポインタと木の**葉** (leaf) を成すアトムとから成る構造であり，共有部分木を持つ木とは，部分木が共有されることを許す構造である．再入ポインタの存在は許すが共有ポインタの存在は許さない構造を再入リストと言う．リストは，共有ポインタと再入ポインタを持つことが可能な構造であり，それゆえ，任意の構造を表現できる最も自由な構造であると言える．

2.2 コピー領域

リスト構造のコピーを作成する時，コピーリストを構成するコピーセルは，**自由リスト** (free list)[14]より1セルずつ取り出される．リストのコピーでは，自由リストのことをコピー領域と呼び，自由リストの性質の違いから，(1)**タグなし** (**tag-free**) **自由リスト**，(2)**タグ付き自由リスト**，(3)**連続領域**の3種類に分類している．

コピー領域を連続番地にとった場合には，コピーは連続領域に作成されるため，作成しつつあるコピーと原リスト構造との区別が容易につくだけでなく，原リスト構造をたどった順序が連続領域に保存される．このため，連続領域へのコピーの場合には，タグなし自由リストやタグ付き自由リストへのコピーの場合よりも高速な処理が実現できる．

タグなし自由リストから取り出されたコピーセルにはタグビットは付いておらず，また，原リストを構成する原セルにもタグビットは付いていない．その一方，タグ付き自由リストへのコピーの場合には，コピーセルと原セルの双方に1ビットのタグビットが付いている．このため，タグ付き自由リストへのコピーの場合には，タグビットの情報を利用してタグなし自由リストへのコピーよりも高速な処理を実現できることがある．

これらのことから，コピー領域として種々の情報の利用しにくいタグなし自由リストへのコピーアルゴリズムは汎用のコピーアルゴリズムとなる．

3. Robsonの汎用リストコピーアルゴリズム

“リストのコピー”と“リストの移動[2]”との違いは，“リストの移動”ではアルゴリズムの実行が終了した時，アルゴリズムへの入力である原リストが破壊され，原リストのコピーだけが原リストの構造を保っているのに対し，“リストのコピー”では原リストが破壊されることはない．本節では，リストをタグなし自由リストへコピーする，すなわちリストの汎用コピーアルゴリズムとして知られるRobsonのリストコピーアルゴリズム[15]を例にとり，“リストのコピー”を作成する上で基本となる事柄を述べる．

一般にリスト構造のコピーには次の過程が必要とされる．(1)共有ポインタや再入ポインタによって指されている部分リストを重複してコピーするのを避けるため，原リストの極大木を決定する．(2)原リストを構成するすべての原セルに関し，各原セルに対応するコピーセルをコピー領域から取り出し，原セルとコピーセルの対を作成する．(3)原セルとコピーセルの対に関し，コピーセルの car, cdr ポインタを求めるため，原セルの car, cdr が指す原セルを求め，それと対を成すコピーセルをそれぞれ copycar, copycdr とし，先のコピーセルの car, cdr に格納する．そして，この時点で原セルとコピーセルの対は分離される．

Robson のアルゴリズムは2パスから成る固定作業領域アルゴリズムであり，パス1，パス2ともに根か

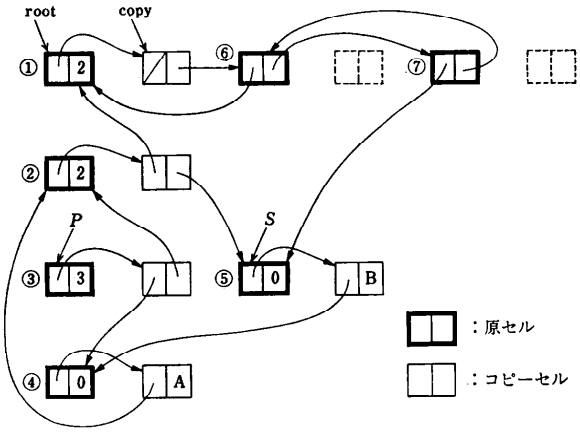

図-2 Robson のアルゴリズム——パス 1——
（セル⑤の処理を終え，セル③の処理に移る状態を示す．
P, *S* はプログラム変数）

ら出発して**ポインタ反転法**[12),16)]によってリストをたどる．また，そのたどり方は，パス 1 では root, car, cdr とパス 2 ではパス 1 とは逆に root, cdr, car とリストをたどる．

図-1 のリストをコピーする場合を考えると，パス 1 では，原リストは **図-2** に示すように，隣接する原セルを指す変数 *P*, *S* を用いてポインタ反転法により，①②③④③⑤③②①⑥⑦⑥①とたどられる．原リストをたどりながら，未だたどっていない原セルに出会うたびに，タグなし自由リストよりコピーセルを取り出し，原セルとコピーセルの対を作成する．したがって，原セルとコピーセルの対は，図-2 の番号①②③④⑤⑥⑦の順に前順序で作成される．この時，原セルの car には，対応するコピーセルを指す先導リンク (forward link) と呼ぶポインタを格納し，コピーセルと原セルを結合する．原セルの car には，対の作成が完了していることを示すマークを入れる．この結果，原リストを葉の方向へたどって行った時，出会った原セルの cdr にマークがあるならば，その原セルに至るポインタは共有ポインタあるいは再入ポインタであると分かる．そのため，原リストの極大木を正確にたどることが可能となるので，任意の原セルに関して，それに対応するコピーセルを重複して自由リストから取り出すことは防止される．

Robson のアルゴリズムの巧妙な点は，マークに値を持たせた点にある．マークは 4 つの値 0, 1, 2, 3 のいずれかをとり，この値によってパス 1 およびパス 2 でポインタ反転法によるリストのたどりを制御している．マークの値は，まず，ポインタの種類に対応して 0, 1, 2 から成る．値 0 は共有ポインタ，再入ポインタあるいはアトムポインタのいずれかを示す．値 1 と 2 は，それぞれ cdr 方向と car 方向を指す F ポインタを示す．各原セルのマークの初期値を 0 とし，パス 1 で原リストをたどりながら，car 方向の F ポインタをたどる時には 2 を，cdr 方向の F ポインタをたどる時には 1 を，それぞれマークに加算する．このことから，ポインタ反転法によって極大木の葉から根に向かって戻る途中でマーク 0 か 2 を持つ原セルに出会ったならば，その原セルはまだ car 方向の処理しか済んでいないことが分かるため，cdr 方向の処理に転ずることが可能となる．したがって，car, cdr ともに F ポインタを持つ原セルに関しては，ポインタ反転法によってパス 1 で 3 回アクセスする間に，**図-3**(a) に示すように，1 回目のアクセス後の状態 II ではマーク 2 を持ち，2 回目のアクセス後の状態 III ではマーク 3 を持つことになる．このようにして，パス 1 終了時には，car が F ポインタでなく，cdr が F ポインタである原セルのマークは 1 となり，また，極大木の葉を形成する原セルのマークは 0 となる．

パス 2 では，原セルとコピーセルの対から成るリストを根を出発点としてポインタ反転法で root, cdr, car とたどりながら，その対が持つマークの値にしたがって対を分離する．この時，原セルの car, cdr には

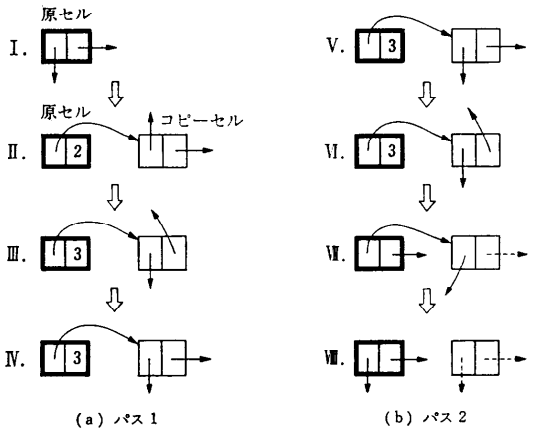

図-3 セルの状態遷移
（上向きポインタは根の方向を，下向き，右向きのポインタは car 方向と cdr 方向を指す．実線は原ポインタ，破線はコピーポインタを示す．）

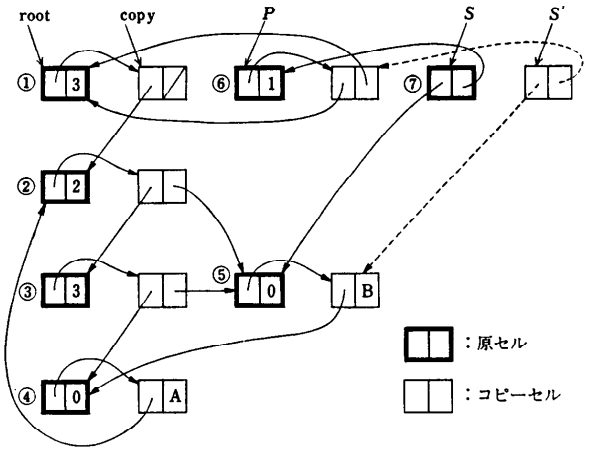

図-4 Robson のアルゴリズム——パス 2——
(セル⑦の処理を終え，セル⑥の処理に移る状態を示す．
P, S, S' はプログラム変数)

元のポインタ（原ポインタ）の値を，そしてコピーセルの car, cdr には，原ポインタが指す原セルと対をなすコピーセルを指すポインタあるいはアトムポインタを格納する．

共有ポインタや再入ポインタが指している原セルとコピーセルの対の分離は，それらをポインタとして持つ原セルとコピーセルの対の分離よりも後に行わなくてはならない．そのため，パス2でのリストのたどりは，パス1とは反対に，**図-4** に示すように，ポインタ反転法により，①⑥⑦⑥①②③⑤③④③②①とたどられ，原セルとコピーセルの対の分離は，⑦⑥⑤④③②①の順に**逆前順序** (reverse preorder) に行われる．

マーク2を持つセルの対では，マークの値から，car ポインタはFポインタであり，cdr ポインタは共有ポインタ，再入ポインタあるいはアトムポインタのいずれかであると分かる．コピーセルの cdr に入る最終値 copycdr は，cdr ポインタがアトムポインタである場合には，コピーセルの cdr に同じアトムを指すポインタを格納すればよい．cdr ポインタが共有ポインタか再入ポインタである場合には，cdr ポインタが指している原セルとコピーセルの対は，リストが逆前順序でたどられているためにまだ分離されていない．それゆえ cdr ポインタが指す原セルの先導リンクをたどり，その原セルに対応するコピーセルを求めることができ，これが共有ポインタか再入ポインタである cdr ポインタの copycdr の値となる．

パス1では，マークの値によってリストのたどりを制御した．パス2では，マークの値とマークの値の有無によりリストのたどりを制御している．car, cdr ともにFポインタを持つセルでは，**図-3** (b)に示すように，パス2開始時のマークの値は3である．根から葉の方向へ進む場合には，状態Vに出会うので，マークの値3から，まず cdr 方向に進むことができる．また，葉から根の方向へ戻っている時には状態VIとVIIの2つの場合があるが，状態VIではマークの値3から car 方向へ進むことができ，状態VIIではマークが無いことから，根の方向への戻りを続けることになる．

ここでは，Robson のアルゴリズムの考え方を述べるにとどめる．詳細なアルゴリズムについては文献 15) を参照されたい．

4. 種々のコピーアルゴリズム

2節で述べたコピーが作成される領域とコピーする対象に従い，代表的なコピーアルゴリズムを **表-1** に示す．表-1 では，コピーする対象は，木，再入リスト，リストの順に分類され，そのコピーアルゴリズムはコピーする対象の複雑さを反映して，この順に処理速度は遅くなる．先に述べたリスト構造の定義にあった共有部分木を持つ木については，そのコピーの作成にはつねにリストのコピーの場合と等しい処理時間がかかるため，リストとの区別はなくなってしまう．

表-1 のアルゴリズムはすべてスタックなどの補助作業領域を使用しない固定作業領域アルゴリズムである．そのため，アルゴリズム中で使用する変数が占める領域を除いては，コピーする対象とそのコピーが占める記憶領域以外には一切の記憶領域を必要としない．どのアルゴリズムも，コピーする対象を構成するセル数 n に対して，$O(n)$ の処理時間を持つが，連続領域へのコピーが最も速く，また，その中でも構造が単純な木のコピーが一番速い．表-1 の分類の各項の中

表-1 リストコピーアルゴリズムの分類

コピー対象＼コピー領域	タグなし自由リスト	タグ付き自由リスト	連続領域
木	Lindstrom[12] Lee[10] 長谷川[7]		Clark[1] 長谷川[7]
再入リスト		長谷川[6]	長谷川[6]
リスト	Robson[16] 長谷川[8]	Lindstrom[13] 長谷川[5]	Fisher[4] Clark[3] 長谷川[5]

では，下に記載されているものほど高速な処理を実現している．

リスト構造のコピーアルゴリズムでは，コピーの作成中に原セルあるいはコピーセルにスタックを埋め込むことができる場合があり[1),3),5),7)]，それによって補助スタックを用いる場合よりも速いだけでなく，より簡潔なアルゴリズムとなっている．木のコピーは，どのコピー領域においても1パスで行われる．再入リストのコピーでは，補助スタックを用いるとアルゴリズムは必然的に2パスとなってしまうが，ポインタ反転法を用いれば1パスのアルゴリズム[6)]となり，前者より高速となる．タグなし自由リストへのリストコピーアルゴリズムである Robson のアルゴリズムも，3節で述べたようにポインタ反転法を基本にしているが，スタックをリストに埋め込んだ，より速いアルゴリズムも発表されている[8)]．

5. あとがき

コピーアルゴリズムの適用例は多い．例えば，不要になったセルを回収するための**回収再生法** (garbage collection) では，リストマーキングによる方法と並んで，必要なリスト構造のコピーを主体として行う方法がある．また，今後，並列処理の比重が増すにつれ，処理対象をコピーし，それらに対して同時に種々の処理を行う場面が増加してくるであろう．

本稿で述べたコピーアルゴリズムだけでなく，リストマーキングアルゴリズムとして有名な Schorr-Waite アルゴリズム[16)]などの固定作業領域アルゴリズムは，ポインタをつなぎ変えてゆく操作から成る**副作用** (side effect) を主体としたアルゴリズムである．そのため，プログラムの正当性を証明する格好の対象となっている[11)]が，十分満足のゆく証明法が確立されてはいない．今後の研究課題であろう．

参考文献

1) Clark, D. W.: A Fast Algorithm for Copying Binary Trees, Inf. Process. Letters, Vol. 4, No. 3, pp. 62-63 (1975).
2) Clark, D. W.: An Efficient List-Moving Algorithm Using Constant Workspace, Comm. ACM, Vol. 19, No. 6, pp. 352-354 (1976).
3) Clark, D. W.: A Fast Algorithm for Copying List Structures, Comm. ACM, Vol. 21, No. 5, pp. 351-357 (1978).
4) Fisher, D. A.: Copying Cyclic List Structures in Linear Time Using Bounded Workspace, Comm. ACM, Vol. 18, No. 5, pp. 251-252 (1975).
5) 長谷川洋：高速リスト・コピー・アルゴリズム，信学技報，AL Vol. 78, No. 233 (1979).
6) 長谷川洋：固定作業領域による再入リストの高速コピー・アルゴリズム，信学技報，AL Vol. 79, No. 92 (1979).
7) 長谷川洋：固定作業領域による木のコピーアルゴリズム，電子通信学会論文誌，Vol. J64-D, No. 7, pp. 585-592 (1981).
8) 長谷川洋：リストをコピーする高速汎用アルゴリズム，信学技報，AL Vol. 82, No. 88 (1983).
9) Knuth, D. E.: The Art of Computer Programming, Vol. 1-Fundamental Algorithms, Addison-Wesley, Mass. (1968).
10) Lee, K. P.: A Linear Algorithm for Copying Binary Trees Using Bounded Workspace, Comm. ACM, Vol. 23, No. 3, pp. 159-162 (1980).
11) 例えば，Lee, S. et al.: The Evolution of List-Copying Algorithms and The Need for Structured Program Verification, Proc. of 6th POPL, pp. 53-67 (1979).
12) Lindstrom, G.: Scanning List Structures without Stacks or Tag Bits, Inf. Process. Letters, Vol. 2, No. 2, pp. 47-51 (1973).
13) Lindstrom, G.: Copying List Structures Using Bounded Workspace, Comm. ACM, Vol. 17, No. 4, pp. 198-202 (1974).
14) McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine-I, Comm. ACM, Vol. 3, No. 4, pp. 184-195 (1960).
15) Robson, J. M.: A Bounded Storage Algorithm for Copying Cyclic Structures, Comm. ACM, Vol. 20, No. 6, pp. 431-433 (1977).
16) Schorr, H. and Waite, W.: An Efficient Machine-independent Procedure for Garbage Collection in Various List Structures, Comm. ACM, Vol. 10, No. 8, pp. 501-506 (1967).
17) Suzuki, N.: Analysis of Pointer "Rotation", Comm. ACM, Vol. 25, No. 5, pp. 330-335 (1982).

(昭和58年12月20日受付)