リスト構造の切り分けと圧縮を行う逐次型 PROLOG 処理系の構造コピー方式†

阿部倫之†† 加久間勝†††

本論文は，リスト構造化したプログラムを処理する構造コピー方式に基づく PLOLOG 処理系において，コピー量の削減と実行時間の短縮を図った新しい構造コピー方式の提案を行ったものである．この新しい構造コピー方式では，リスト構造を動的に切り分ける方法と CDR コーディングによりリスト構造を圧縮する方法を導入している．そこでは，リスト構造の静的分類による切り分けの効率化と CDR コーディングのための最適処理についての提案も行っている．また，その効果を確認するために，インタプリタ型の処理系を試作して定量的な評価を行っている．その結果，次のことが得られた．(1)構造の切り分けと最適化 CDR コーディングを融合させることにより，コピー量の大幅な削減が可能である．特に，構造の切り分けにより動的 CDR コーディングの可能性を増加できる．(2)構造の切り分けにより生じるオーバヘッドは，構造を静的に分類することにより抑制できる．(3) CDR コーディングにより生じるオーバヘッドは，構造の切り分けを行うことにより吸収することが可能である．また，削減されるコピー量が多いほど実行時間の短縮効果が期待できる．

1. はじめに

論理型言語 PROLOG は可読性，検証性のよさと単一化，後戻り機能を持つ言語として，人工知能システムなどの記述に多用されている．この PROLOG を逐次実行する場合，後戻り機能を実現するために節の選択点 (choice point) に関する情報や呼び出し元であるゴールの環境（変数の束縛情報）を保持しながら処理を進める必要があるためメモリを多量に使用する．そこで，保持する必要のなくなったローカル情報を動的に削除していくなどの最適化が行われている[8]．ここで，WAM (Warren Abstract Machine)[4] に基づくコンパイラ型処理系（以下，コンパイラ）では，項レベル具体値 (instance) の表現方法に構造コピー方式[2]を用いており，メモリ管理が容易なことからインタプリタ型の処理系[5]（以下，インタプリタ）でも採用している．構造コピー方式は，変数に項を束縛するとき項のコピーを行うため，コピーを行わない方式（構造共有方式）[1] に比べてメモリを多く使用する．また，コピーされた項は，バックトラックのときにしか削除されないグローバル情報であるため，スタック中に存在している期間が長い．この性質は，スタックの使用量が増加する要因となっているため，コピー量を減少させるための最適化方法の検討が望まれる．

PROLOG プログラム（以下，プログラム）の内部形式[7]（メモリ上の表現形式）は，レコード形式とリスト形式が知られており，多くの PROLOG 処理系（以下，処理系）では，レコード形式を採用している．レコード形式はリスト形式よりもプログラムをコンパクトに表現できるため記憶効率がよい．しかし，リスト形式は

(i) 処理系の記述が簡潔になる
(ii) プログラムの追加，削除が容易にできる
(iii) ガーベジコレクションによる使用済み領域の再使用を中心としたメモリ管理が容易である

の点でレコード形式よりも有利である．すなわち，リスト形式は処理系に対して高い操作性を提供する．このリスト形式と構造コピー方式を併用した場合に発生する主な問題点は，コピー量の増加である．本論文では，リスト形式を用いている処理系（リスト構造化されたプログラムを扱う処理系）が構造コピー方式を採用する場合の最適化について述べる．

コピー量を減少させる最適化の例として文献 10) がある．これは WAM を対象としたものであり，コピー領域がスタック型の連続した領域となっていることに着目して，複合項であるリストを構成するセル（リストセル）の CDR 部を動的に省略 (CDR コーディング) してコピー量を少なくしている．しかし，CDR コーディングに伴う処理の変更は最小限にとどめており，CDR コーディングの効果を増加させるための方法については未検討である．本論文では，WAM の枠組みにとらわれずに構造コピー方式の基

† An Efficient Structure Copying Method for Sequential PROLOG Processing System Using List Structure Classification and CDR-Coding by NORIYUKI ABE (Totsuka Works, Hitachi, Ltd.) and MASARU KAKUMA (Department of Information Engineering, Faculty of Engineering, Kanazawa Institute of Technology).

†† 日立製作所(株)戸塚工場
††† 金沢工業大学工学部情報工学科

本から出発して,

(1) リスト構造の静的分類による動的切り分け

(2) CDR コーディングによるリスト構造の圧縮を導入することでコピー量を減少させる新しい構造コピー方式を提案する.

2. プログラムの構成と構造コピー方式

2.1 プログラムの構成

プログラムの内部形式は,

(i) 項の構造体イメージを完全に持つ

(ii) 項の構造体イメージを部分的に持つ

(iii) 項の構造体イメージを持たない

場合が考えられ,処理の形態はそれぞれの場合で異なる.一般に,(i)のタイプはインタプリタであり,(ii)と(iii)のタイプはコンパイラである.(ii)に属するコンパイラとして PLM[3] (構造共有方式を採用している),(iii)に属するものとして WAM がある.このとき,プログラムをリスト形式で表現した場合,項のコピー量に直接影響を与えるのは(i)と(ii)である.したがって,コピーの最適化方法 (以下,最適化) の検討は,プログラムが項の構造体イメージを持っている場合を中心に行う.特に,リスト構造 (以下,構造) の切り分けは,(i)と(ii)のみを対象としたものである.

プログラムを構成するセルは,PSI[6] などのタグアーキテクチャで採用しているタグ付きデータとする.これをデータセルと呼ぶ.リストセルはデータセルを二つ使用して構成する (**図 1**).このリストセルの構成法は,後で述べる CDR コーディングを行う上で都合がよい.

2.2 構造コピー方式

構造コピー方式は,変数に値が束縛されたことによって具体化した項を表現するための一方法である.項のコピーは変数に項を束縛するときに行われ,コピーした項に存在する変数は,値を格納するセルからポインタで参照される.このセルを変数セルと呼び,コピーした項中の変数に対する参照ポインタをダイレクト・ポインタ (DP) と呼ぶ.この例を**図 2** (a)に示す.図中の 'VAR' は変数,'DP' はダイレクトポインタを表すタグである.変数セルは,一つのデータセルで構成してスタックに格納する.また,コピー領域としてコピー専用のスタック (コピースタック) を設ける.ここで,項をコピーするとき同じ変数が複数存在する場合は,それぞれ同じ値が束縛されるように処理する必要がある.WAM では,二番目以降の変数を変数セルの内容に書き換えることでこの問題を解決している (図2(b)).

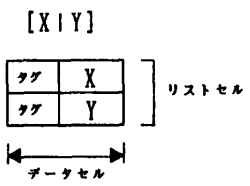
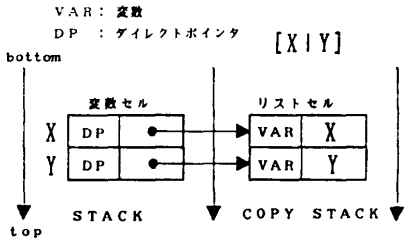


図 1 リストセルの構成
Fig. 1 Constitution of list cell.

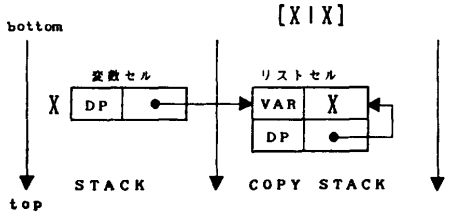
3. リスト構造の動的切り分け

項をコピーする目的は項を直接書き換えるためであるから,書き換えられる可能性のない項は本来コピーする必要はない.そこで,変数を含まない項を切り分けて,その項をコピーしないようにすることにより,コピー量を減少できる.この例を**図 3** (a)に示す.図中の 'CON' は定数,'STR' は構造を参照するポインタを表すタグである.この最適化は,項の構造体イメージをプログラムが持っている場合に有効である.

項はリスト形式に変換されているため,切り分けそのものは自然な形で行うことができる.しかし,コピーする構造中に変数が含まれているか否かの識別を,単一化 (unification) の際に動的に行った場合,オーバヘッドが発生する要因となる.そこで,項をリスト形式に変換 (コンパイル) するときに構造の分類[9]



(a) コピーする構造に同一の変数がない場合



(b) コピーする構造に同一の変数がある場合

図 2 構造コピー方式
Fig. 2 Structure copying method.

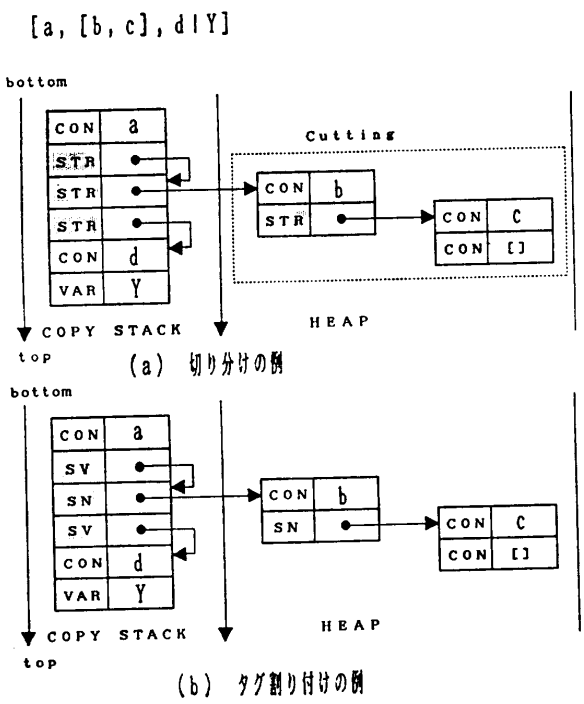


図 3 リスト構造の切り分けとタグ割り付け
Fig. 3 The cutting off list structure and tag allocation.

を行って，結果をタグとしてデータセルに付加することを考案した．このタグ割り付けの例を図3(b)に示す．識別の対象となるのは構造を持つ項であるから，構造を参照するポインタに付加するタグを，変数を含む構造を参照するポインタとそうでないものとに分ける．このとき，ポインタの種類が複数ある場合，それぞれについて同様のことを行う．図3(a)では，構造を参照するポインタを一種類('STR')のみにしているため，新しいタグは二つあればよい．図3(b)では，変数を含む構造へのポインタに 'SV'，変数を含まない構造へのポインタに 'SN' を割り付けている．このタグ割り付けにより，構造の切り分けを伴ったコピーを

(i) データセルがポインタでない場合，そのデータセルを含むリストセルのコピーを行う．

(ii) データセルがポインタの場合，そのデータセルを含むリストセルのコピーを行い，タグが 'SV' ならばポインタの参照している先をコピーの対象とするが，タグが 'SN' ならばなにもしない．

の簡単な手続きで実現できる．また，切り分けそのものは高々一回のタグ判定で可能になる．

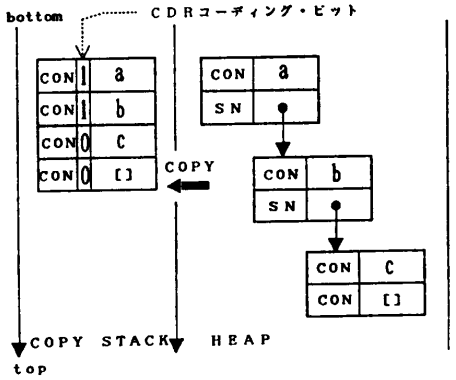


図 4 静的 CDR コーディング
Fig. 4 Static CDR-coding.

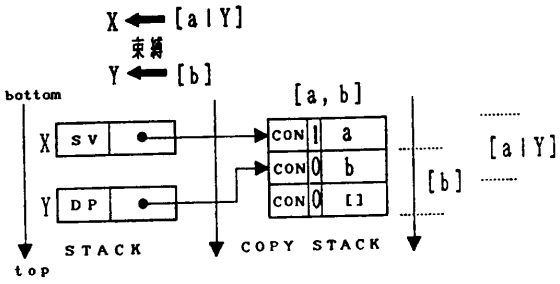


図 5 動的 CDR コーディング
Fig. 5 Dynamic CDR-coding.

4. リスト構造の圧縮

4.1 CDR コーディング

CDR コーディングとは，構造の格納先が連続した領域のとき，リストセルの CDR を省略して格納する構造の圧縮技法である(**図 4**)．このとき，CDR コーディングの有無を表すビット (CDR コーディングされているときビット '1' を格納する) を CAR に設ける．これを CDR コーディング・ビットと呼ぶ．PROLOG 処理系の場合コピー先がスタックであるため，一括してコピーされる構造は必ず CDR コーディングできる．これを静的 CDR コーディングと呼ぶ．PROLOG によるリストの結合は，節を呼び出すときの単一化処理を利用して行うことができる．このとき，リストセル中の変数に，結合すべきリストセルを直接束縛 (変数をリストセルを参照するポインタに書き換える) して新しい構造を作るが，この束縛の際に CDR コーディングが可能である(**図 5**)．これを動的 CDR コーディングと呼ぶ．WAM においても，リストのスタック中の構造はリスト構造であるため，この二つの CDR コーディングは有効である．

動的 CDR コーディングを実現する場合，変数セル

からの DP による参照とリストセルからの DP による参照の仕方に工夫を要する．

（1） 変数セルからの DP による参照

[a|X] がコピーされており（**図 6**(a)），[Y] が変数 X に束縛される場合，コピースタックは図 6 (b)の状態になる．このとき，変数セル X の DP は [Y] を参照しており，変数セル Y の DP はデータセル Y を参照している*が，そのままではこの識別ができない．そこで，CDR コーディング束縛が行われたとき変数セルに目印を付ける（ビット'1'を格納する）ことでこの識別を行う（図 6 (c)）．この目印を CDR コーディング束縛ビット（以下，束縛ビット）と呼ぶ．このビットをセットする方法は，コピーした構造に対する直接代入が，変数セルの DP を使用する場合と使用しない場合とで異なる．前者の場合は，変数セルの位置を覚えておき，CDR コーディング束縛が確定した後にセットする．後者の場合は，変数セルを介さないで直接構造中の変数に束縛する場合であり，変数セルの位置を得ることができない．そこで，構造をコピーするとき，その中の変数を対応する変数セルへのポインタにあらかじめ書き換えておくことで変数セルの位置が得られるようにする．このポインタを使用して束縛ビットをセットする．

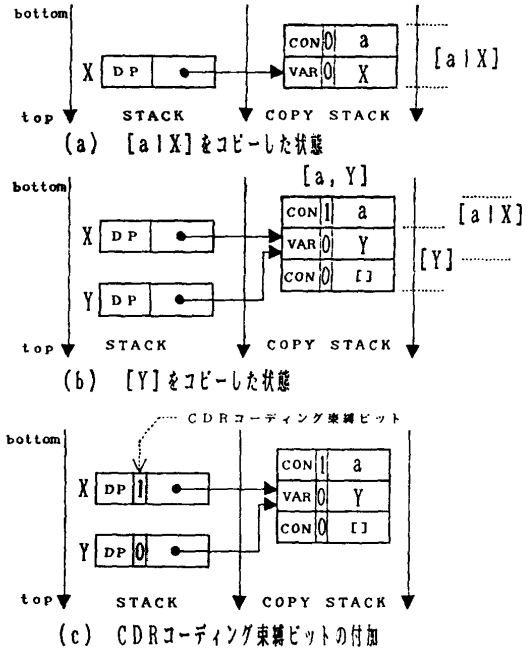


図 6 変数セルからの DP による参照
Fig. 6 Reference by DP (from variable cell).

* この章では，リストセルの CAR と CDR を直接指すときに限りデータセルと呼ぶことにする．例えば，[X|Y] の CAR はデータセル X，CDR はデータセル Y と呼ぶ．

以上により，DP の識別は次のように行う．

（i） DP の参照先の一つ前に格納されているデータセルの CDR コーディング・ビットが'0'のとき，DP の参照先はデータセルである．

（ii） DP の参照先の一つ前に格納されているデータセルの CDR コーディング・ビットが'1'のとき，変数セルの束縛ビットが'1'であるならば DP の参照先はリスト構造である．また，'0'であるならばデータセルである．

（2） リストセルからの DP による参照

構造（リストセル）をコピーするとき，既にコピーした変数が現れた場合，WAM に基づいて先に現れた変数を参照するように構造中の変数を DP に書き換える（図 2 (b)）と，DP の参照先が CDR コーディングされることはないため，DP は必ずデータセルを指している．したがって(1)のような識別は必要としない．しかし，WAM とは逆に後に現れた変数を参照するように変数を DP に書き換える場合（**図 7**）は，DP の参照先が CDR コーディングされる可能性が出てくるため，DP の識別が必要となる．これについては次の節で述べる．

4.2 CDR コーディングのための最適処理

（1） コピーの方法

構造のコピーは，CAR を優先してコピーしていく深さ優先コピーと CDR を優先してコピーしていく幅優先コピーが考えられる（**図 8**）．構造を一括してコピーする場合，深さ優先コピーでは，ネスティングしている構造が上位の構造に編み込まれるため，静的 CDR コーディングが行われない可能性がある．これに対して幅優先コピーでは，常に一定の静的 CDR コーディングが保証されるため，深さ優先コピーよりも有利である．動的 CDR コーディングの場合は，他の単一化処理の影響をうけるため，有利性を一意に決定できない．実行時間は，コピー量が同じ場合には差がないため，幅優先コピーの方が CDR コーディング向きといえる．これにより静的 CDR コーディングの

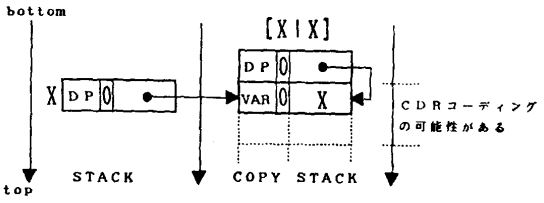


図 7 DP の割り付け方法
Fig. 7 A method of DP allocation.

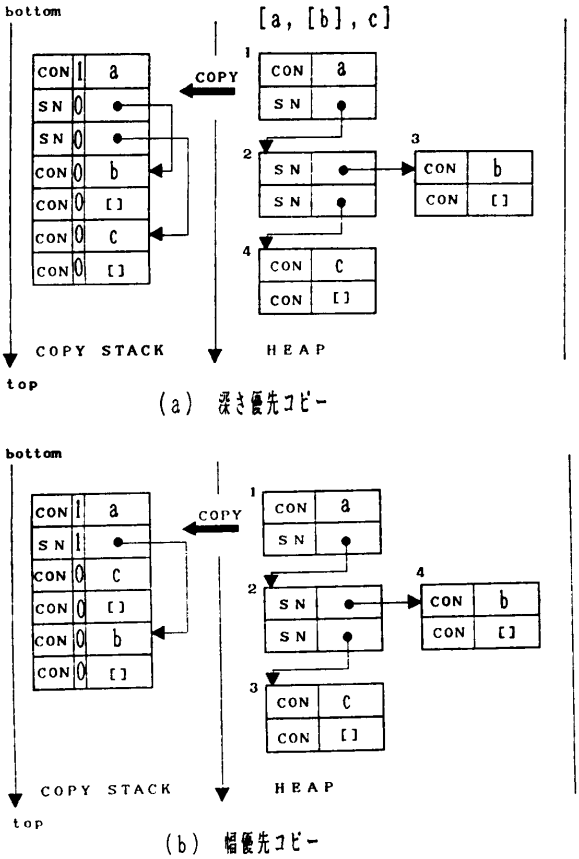


図 8 コピーの方法
Fig. 8 Copying method.

可能性が増加する.

(2) DP 処理の変更

WAM では，構造（リストセル）をコピーするとき，既にコピーした変数が現れた場合，先に現れた変数を参照するように変数を DP に書き換える．ここで，この変数が最後の CDR にある場合 CDR コーディングできなくなる．図 9(a)は $[X|X]$ がコピーされ，X に $[Y]$ が束縛され，$[Z]$ がコピーされて，$[Y]$ と $[Z]$ の単一化が行われたときのスタックの状態である．このとき $[Y]$ が CDR コーディング束縛されていないのがわかる．ここで，WAM とは逆に，後に現れた変数を参照するように変数を DP に書き換えることを考える．この例を図9(b)に示す．これは図9(a)と同じ例であるが，$[Y]$ が CDR コーディング束縛されている．このとき，データセル X の DP は $[Y]$ を参照しており，データセル Z の DP はデータセル Y を参照している．したがって，DP の識別が必要となる．これには，変数セルと同様に束縛ビットを設ける．このビットには，構造中の変数どうしが単

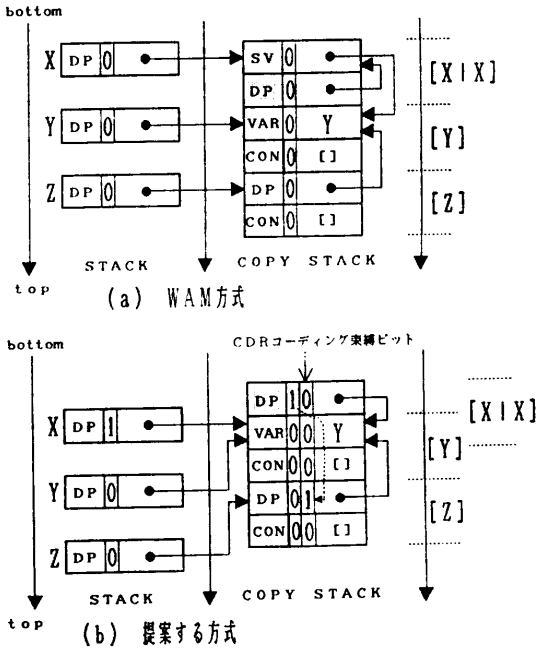


図 9 リストセルからの DP による参照
Fig. 9 Reference by DP (from list cell).

一化されたとき，DP に書き換えられるデータセルに，DP の参照先の一つ前のデータセルの CDR コーディング・ビットと同じものを格納する（図中の破線矢印）．これにより，コピーした構造中の変数どうしの単一化が，CDR コーディング以前であればビット '0' が格納され，CDR コーディング以後であればビット '1' が格納される．

以上により，DP の識別は次のように行う．

(i) DP の参照先の一つ前に格納されているデータセルの CDR コーディング・ビットが '0' のとき，DP の参照先はデータセルである．

(ii) DP の参照先の一つ前に格納されているデータセルの CDR コーディング・ビットが '1' のとき，DP を格納しているデータセルの束縛ビットが '0' であるならば DP の参照先は構造（リストセル）である．また，'1' であるならばデータセルである．

この変更により動的 CDR コーディングの可能性が増加する．

(3) 構造の切り分けを適用

動的 CDR コーディングを行うためには，コピーの位置とそのポインタの格納先が隣接している必要があるため，不要な構造はなるべくコピーしない方がよい．そこで，3章で述べた構造の切り分けを併用することを考える．コピーの方法を幅優先コピーとした場

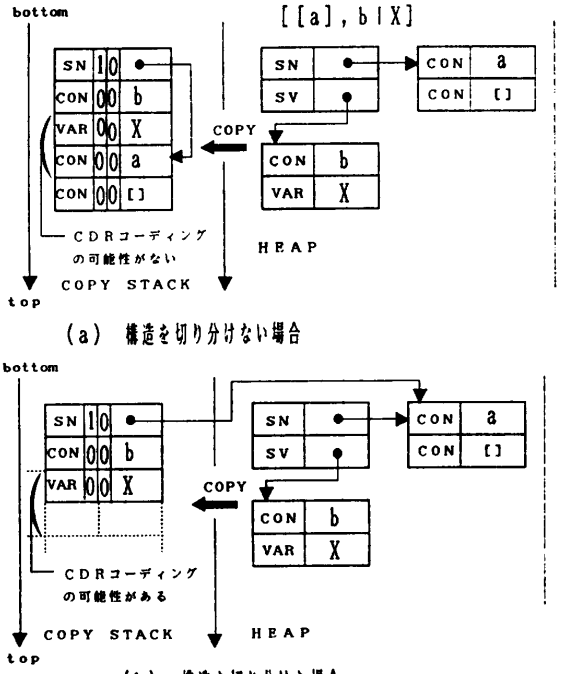


図 10 CDR コーディングとリスト構造の切り分け
Fig. 10 CDR-coding and the cutting off list structure.

合の例を**図 10** に示す．(a)は切り分けない場合であり，(b)は切り分けを行った場合である．(a)では動的 CDR コーディングの可能性はないが，(b)では，最後にコピーの対象となった［a］が切り分けられたため，動的 CDR コーディングが可能な状態となっている．これにより，動的 CDR コーディングの可能性が増加する．

5. 評　　価

5.1 試作処理系と評価プログラム

構造の切り分けと圧縮の効果を定量的に評価するため，次の四種類のインタプリタ型処理系を試作し，三種類の評価プログラムを実行させて，メモリ使用量と実行時間を実測した．

(TYPE 1) 単純コピー
(TYPE 2) 構造の切り分けを伴ったコピー
(TYPE 3) 構造の圧縮を伴ったコピー
(TYPE 4) 構造の切り分けと圧縮を伴ったコピー

コピーの方法は幅優先コピーを採用している．各処理系はC言語で記述しており，プログラムサイズはそれぞれ 40 キロバイト程度で，PC-9801/vm (10 MHz) 上にインプリメントしている．データセルのフォーマットを**図 11** に示す．タグ部が8ビット，データ部が

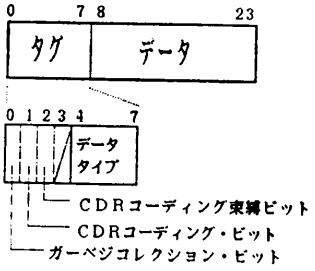


図 11 データセルのフォーマット
Fig. 11 Data cell format.

16ビットの構成であり，タグ部の下位4ビットがデータタイプ，上位4ビットにガーベジコレクション・ビット，CDR コーディング・ビット，CDR コーディング束縛ビットを格納している．

評価プログラムは，データベース操作，Naive_reverse (30要素)，8_Queen (全解探索) である．各プログラムは次の特徴を持つ．

(1) データベース操作は構造の切り分けが多く行える．
(2) Naive_reverse は CDR コーディングが多く行える．
(3) 8_Queen は切り分け，CDR コーディングともあまり行えない．

ここで，使用するデータベースには，大学の開講科目の受講者リストを1科目レコードとして，50レコード登録している．1科目レコードを一つの事実節で表現しているため，データベースは 50 個の事実節より成る小規模なものである．このデータベースの操作は，与えられた科目名リストに記載された全科目を受講している受講者名の検索を行うものである．1科目レコードの構成と操作プログラムをそれぞれ付録1，付録2*に示す．なお，実際の評価は，1科目レコード中の受講者レコード数は一つで，全科目レコードが同じ受講者レコードを持つものに限定して行った．また，科目名リストには全科目名が記載されている．

5.2 結果と考察

(1) メモリ使用量

実測により得られたメモリ使用量を**表 1** に示す．この表から次のことが指摘できる．

(i) 全メモリの減少量とコピースタックの減少量とがほぼ一致することから，メモリの削減効果は，コピースタックの削減のみによるものと考えられる．そ

* このプログラムは，最適化の効果が顕著に現れる一例として作成したものであり，プログラムの最適化は行っていない．例えば，member プログラムはメモリ効率がよくない形式で記述している．

れゆえ以下の考察は，コピースタックのみに着目して述べる．

(ii) TYPE 1 と TYPE 2 の差は構造の切り分けによるコピーの減少効果を，TYPE 1 と TYPE 3 の差は CDR コーディングの効果を示している．また，TYPE 1 と TYPE 4 の差は，上記の 2 効果によるメモリの減少量を示している

(iii) TYPE 3 では，TYPE 2 で切り分けられた構造を約半分のサイズに圧縮してコピーする．したがって，TYPE 4 の減少量は，TYPE 2 の減少量の半分と TYPE 3 の減少量との和にほぼ一致するものと予想できる．Naive_reverse と 8_Queen では予想値に合致しているが，データベース操作ではメモリの減少量が予想値よりも多くなっている．

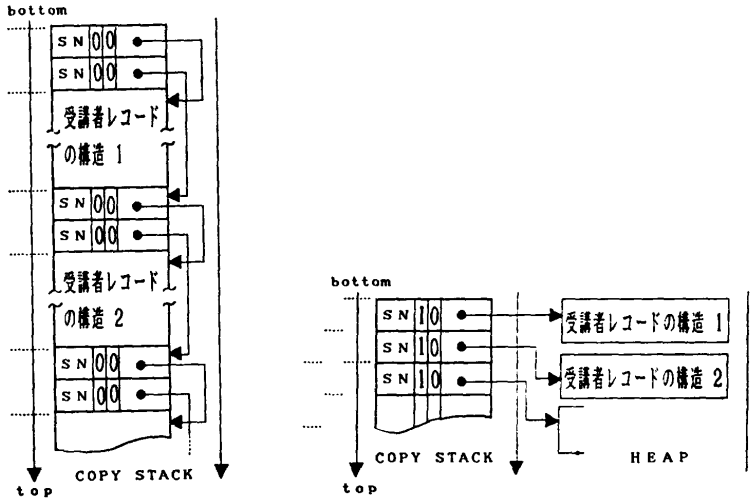


図 12 リスト構造の切り分けと CDR コーディングを併用した場合の効果
Fig. 12 The effect of cases where the system used both the cutting off list structure and CDR-coding in combination.

(iv) この理由は次のように考えられる．データベース操作には，得られた全受講者レコードのリストを構成するステップがあり，このリストを構成するために新しく生成（コピー）されたリストセルにデータベースからコピーした受講者レコードが交互に編み込まれる (**図 12**(a))．ここで，受講者レコードをコピーしなければ，このリストは動的 CDR コーディングを行いつつ構成することができる(図 12 (b))．したがって，TYPE 4 では構造の切り分けにより，TYPE 3 よりも動的 CDR コーディングが多く行われたため，メモリの削減量が増加したものと考えられる．このことは，構造の切り分けが CDR コーディングに対して有効であることを実証している．

(v) メモリの減少率を**表 2** に示す．構造の切り分けによる効果は，データベース操作のときのみ顕著であるが，CDR コーディングではすべての場合について効果を発揮している．データベース操作において，(iv)で述べた動的 CDR コーディングの増加によるコピースタックの減少率は約 14% と考えられる．

(vi) この結果から本論文で提案した方法は，メモリ使用量の減少の面では効果があると考えられる．

(2) 実行時間

実測により得られた実行時間を**表 3** に示す．この表から次のことが指摘できる．

(i) Naive_reverse と 8_Queen は類似の傾向を

表 1 メモリ使用量 (Kbyte)
Table 1 Memory use (Kbyte).

プログラム / 処理系	データベース操作	Naive_reverse	8_Queen (全解探索)
TYPE 1	8.49 (3.29)	13.28 (2.96)	5.92 (0.32)
TYPE 2	6.69 (1.49)	13.1 (2.79)	5.91 (0.31)
TYPE 3	7.44 (2.25)	11.97 (1.66)	5.85 (0.26)
TYPE 4	6.09 (0.9)	11.88 (1.57)	5.84 (0.25)
TYPE 1−TYPE 2	1.8 (1.8)	0.18 (0.17)	0.01 (0.01)
TYPE 1−TYPE 3	1.05 (1.04)	1.31 (1.3)	0.07 (0.06)
TYPE 1−TYPE 4	2.4 (2.39)	1.4 (1.39)	0.08 (0.07)
(TYPE 1−TYPE 2)/2 +(TYPE 1−TYPE 3)	1.95 (1.94)	1.4 (1.39)	0.08 (0.07)

( )内はコピースタックの使用量

表 2 コピースタックの減少率 (%)
Table 2 Rate of decrease in COPY STACK (%).

プログラム / 処理系	データベース操作	Naive_reverse	8_Queen (全解探索)
TYPE 1	0.00	0.00	0.00
TYPE 2	54.71	5.74	3.13
TYPE 3	31.61	43.92	18.75
TYPE 4	72.64	46.96	21.88
(TYPE 2/2)+TYPE 3	58.97	46.79	20.32
TYPE 4− (TYPE 2/2+TYPE 3)	13.67	0.17	1.56

表 3 実行時間 (sec)
Table 3 Execution time (sec).

プログラム / 処理系	データベース操作	Naive_reverse	8_Queen (全解探索)
TYPE 1	2.17 (0.00)	2.36 (0.00)	245.79 (0.00)
TYPE 2	1.52 (−29.95)	2.41 (2.12)	248.53 (1.11)
TYPE 3	2.25 (3.69)	2.47 (4.66)	258.49 (5.17)
TYPE 4	1.6 (−26.27)	2.5 (5.93)	260.69 (6.06)

( )内は実行時間の増加率 (%)

示している.

(a) 構造を切り分けることにより処理量が増加するために, 実行時間は一般に増加する. このオーバヘッドは, TYPE 2 の結果より 1%〜2% と推定されるが, 構造の静的分類の効果によって低く抑えられている.

(b) CDRコーディングを行うことにより, (a)と同様に実行時間が増加する. このオーバヘッドは, TYPE 3 の結果より 4%〜5% と推定される.

(c) 構造の切り分けと CDR コーディングを同時に行うことにより, (a)や(b)と同様に実行時間が増加する. このオーバヘッドは, TYPE 4 の結果より約 6% と推定される.

(ii) データベース操作の場合には, 実行時間は逆に大幅に減少している. この理由は次のように考えられる.

(a) TYPE 2 は, コピー量が大幅に削減されたことにより, 構造を切り分けることによる処理量のオーバヘッドを吸収し, かつ実行時間をさらに短縮できたことを示しており, 約 30% の実行時間の減少が得られている.

(b) TYPE 3 は, 8_Queen 等の場合と同様に CDR コーディングによるオーバヘッドを示しており, 約 4% の実行時間の増加率になっている.

(c) TYPE 4 は, TYPE 2 と TYPE 3 の効果によるものであり, 約 26% の実行時間の減少率となっており, TYPE 2 と TYPE 3 の効果の和になっている.

(iii) (i)と(ii)で示したオーバヘッドは, タグのビット操作によるデータタイプと CDR コーディング(束縛)ビットの抽出, 設定およびその判定に起因したものといえる. 特に, 処理系を高水準言語 (C言語) でインプリメントしたことによって, オーバヘッドの発生がより表面化したものと考えられる. これらのオーバヘッドは, ビット操作専用の命令を用意したり, タグの判定を他の処理と並行して行う機能などをハードウェアに付加することにより減少できるものと考える.

6. おわりに

本論文では, リスト構造化されたプログラムを持つ処理系を対象として, 構造の切り分けと CDR コーディングによる圧縮を行うことでコピー量を減少させる構造コピー方式を検討した. また, その効果を確認するためインタプリタ型処理系の試作と評価を行い,

(1) 構造の切り分けと最適化 CDR コーディングを融合させることにより, コピー量の大幅な削減が可能である. 特に, 構造の切り分けにより動的 CDR コーディングの可能性を増加できる.

(2) 構造の切り分けにより生じるオーバヘッドは, 構造を静的に分類することにより抑制できる.

(3) CDR コーディングにより生じるオーバヘッドは, 構造の切り分けを行うことにより吸収することが可能である. また, 削減されるコピー量が多いほど実行時間の短縮効果が期待できる.

が得られた.

今後は, ガーベジコレクションおよび WAM などのコンパイラ型処理系への適用を含んだ総合的な検討を行う必要があるものと考える.

参考文献

1) Mellish, C. S.: An Alternative to Structure Sharing in the Implementation of a Prolog Interpreter, in *Logic Programming* (eds. Clark, K. L. and Tarnlund, S. A.), pp. 99-106, Academic Press, New York (1982).
2) Bruynooghe, M.: The Memory Management of Prolog Implementation, in *Logic Programming* (eds. Clark, K. L. and Tarnlund, S. A.), pp. 83-98, Academic Press, New York (1982).
3) Warren, D. H. D.: Applied Logic—Its Use and Implementation as a Programming Tool, Technical Note 290, SRI International (June 1983).
4) Warren, D. H. D.: An Abstract Prolog Instructon Set, Technical Note 309, SRI International (Oct. 1983).
5) van Emden, M. H.: An Interpreting Algorithm for Prolog Programs, in *Implementations of PROLOG* (ed. Campbell, J. A.), pp. 93-110, Ellis Horwood (1984).

6) 山本，横田，西川，瀧：逐次型推論マシンΨのマイクロインタプリタ，ICOT, Technical Report : TR-079 (Sep. 1984).
7) 中村：Prolog処理系，情報処理，Vol. 25, No. 12, pp. 1329-1335 (Dec. 1984).
8) Warren, D. H. D. : Optimizing Tail Recursion in Prolog, in *Logic Programming and Its Applications* (eds. van Caneghen, M. and Warren, D. H. D.), pp. 77-90, Ablex Publishing Corporation Norwood, New Jersey (1985).
9) 阿部，加久間：可変項を具体化の対象とするProlog処理系の一検討，第36回情報処理学会全国大会論文集，6 H-3, pp. 795-796 (1988. 3).
10) 砥崎，松本，上原，豊田：PROLOG処理系アーキテクチャの拡張と最適化方式の提案，*Proceedings of the Logic Programming Conference '88*, pp. 151-160 (1988).

```
course ( <科目名>,
         attendant_list ( [
                          [<受講者番号>,   ↑
                           <受講者名>,     |  受講者レコード
                           <出席回数>,     |
                           <成績> ],       ↓
                          ...
                         ]
                        )
       ).
```

付録 1 科目レコードのフォーマット
Appendix 1 Course record format.

```
member(X,[X|_]).
member(X,[_|Y]):-member(X,Y).
search_course([],[]).
search_course([X|Y],[H|Z]):-
              course(X,attendant_list(H)),
              search_course(Y,Z).
gen_mem(_,[]).
gen_mem([_,U,_,_],[X|Y]):-
        member([_,U,_,_],X),
        !,
        gen_mem([_,U,_,_],Y).
com_member(_,[]).
com_member(E,CL):-
          search_course(CL,[L1|L2]),
          member([_,E,_,_],L1),
          gen_mem([_,E,_,_],L2).

?-com_member(COM,[<科目名>,...]).
```

付録 2 データベース操作プログラム
Appendix 2 Data base operation program.

(昭和63年8月4日受付)
(平成元年5月9日採録)

阿部 倫之（正会員）

昭和36年生．昭和59年金沢工業大学情報処理工学科卒業，昭和61年同大学大学院情報工学専攻修士課程修了，平成元年同大学院情報工学専攻修士課程満期退学，同年日立製作所入社，現在同社戸塚工場に勤務．論理型言語とその処理系および人工知能，知識工学などに興味を持つ．電子情報通信学会，人工知能学会各会員．

加久間 勝（正会員）

大正14年生．昭和22年浜松工業専門学校電気通信科卒業，同年，逓信省入所，電電公社武蔵野電気通信研究所基幹交換研究部処理プログラム研究室長を経て，昭和53年金沢工業大学情報工学科教授，現在に至る．工学博士．複合計算機システム，オペレーティング・システム，知識工学，論理プログラミング言語の研究に従事．著書「システム工学」（共著 オーム社）．電子情報通信学会会員．