Vol. 27 No. 7 情報処理学会論文誌 July 1986

Prologシステムのための Binary Tree Memory†

楊 容†† 正畑康郎†† 相磯秀夫††

本論文では，二進木データを効率よくメモリに格納し処理するための Binary Tree Memory (BTM) を提案する．これは，二進木データの構造をポインタを用いずに表現して二進木データの要求するメモリ量や二進木データへのアクセス量を削減することにより，二進木データ処理の効率を向上させようとするものである．ポインタを用いずに構造を表現するために，注目するノードのアドレスを計算する BTM 関数を使用する．次に，本論文では，BTM を Prolog 処理系に用いた際の効果について評価を行う．BTM を Prolog のプログラムを蓄えるプログラム領域に適用すると，DEC-10 Prolog と比較して，メモリ要求量を 20～48% 削減することができ，プログラム領域へのアクセス量を 17～36% 減少させることができる．

1. はじめに

コンピュータの応用分野は従来の数値処理の分野から，人工知能，知識情報処理といった非数値処理を指向した分野へ急速に広がりつつある．非数値処理のアプリケーションは，木に代表される非定型構造データを主な演算対象とする．この構造は数値処理でよく用いられる，配列に代表される定型構造データとは異なるため，既存のアーキテクチャでは効率的に扱うことが難しく，非数値処理向きの専用マシンの開発が必要である．本論文では，この非数値処理を指向したマシンの研究分野における一つの基本的な課題，すなわち，木構造データの高速処理について，Prolog システムにおける木構造データを取り挙げて検討する．まず，木構造データを効率よく処理するための Binary Tree Memory (以下 BTM と呼ぶ) というメモリアーキテクチャを提案し，次に，BTM を Prolog システムに用いた際の効果についての評価を行う．

2. Binary Tree Memory (BTM)

2.1 背　景

前章に述べた非数値処理分野の主な演算対象は，Prolog では複合項，またCでは構造体，Pascal ではレコードと呼ばれる．このような構造データは，要素の型が同一である必要はなく，しかも，要素の再帰的定義も許すという特徴を持つ．このため一般的には，このようなデータは木構造を用いて表現される．図1に示すように，線形アドレス空間を持つ既存のメモリ上では，木構造はポインタを用いることによって表現される．もし，ある要素が，さらに構造を持つならば，ポインタを利用してその構造を指す．木構造をリストの形に変形して，メモリ上に表現することもできる(図 2)．リストの第1要素で関数記号を，第2要素以後でその引数を表すことにすれば，図2(a)に示す構造は図2(b)に示すリストで表現できる．このリストはメモリ上では図2(c)に示すように表現される．

従来の Prolog 処理系ではほとんど図1と図2に示すような二種類の方法のうちの一つ，あるいは双方が組み合わされて使われている．この二種類の方法には共通してポインタが必要である．ポインタを利用すると，任意の場所に置かれているデータを動的に連結することができる．Prolog 処理系では，共有変数や共有構造などを効率よく処理するためには，ポインタは欠かせないものである．しかし，それ以外の用途のポインタの使用は実行効率の低下を招くと我々は考える．例えば，図1，図2の例の場合，ポインタが全データの約 40% を占めてしまうので，メモリ要求量と構造へのアクセス回数の約 40% がポインタのために消費されていると言える．

よって，木構造データのポインタに関する処理の改良により，非数値処理向きマシン全体の効率を向上させることが可能であると考えられる．このためには，よりよい木構造データの表現，あるいは，木構造データのための専用メモリを開発することが必要となる．この考えに基づいて，本論文では BTM という新しいメモリアーキテクチャを提案する．

2.2 BTM の概念

BTM とは，木構造を，ポインタの助けを借りずに記憶できるメモリである．まず，BTM で扱われるデータは二進木であるとする．任意の木構造は適当な

† Binary Tree Memory for Prolog Systems by RONG YANG, YASURO SHOBATAKE and HIDEO AISO (Department of Electrical Engineering, Keio University).

†† 慶応義塾大学理工学部

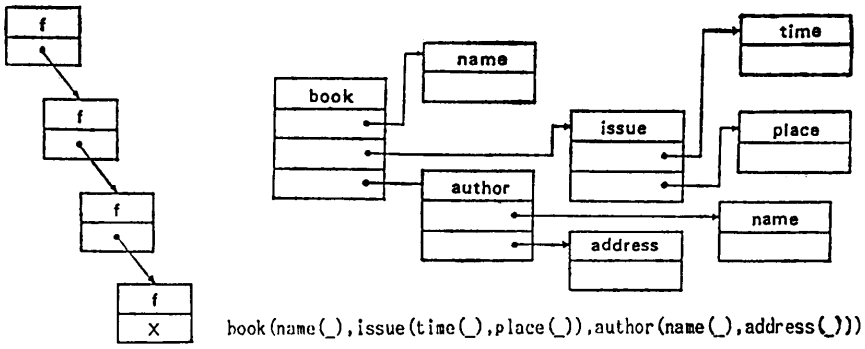

図 1 線形アドレス空間上での木構造の表現 (1)
Fig. 1 Representation of tree structured data (1).

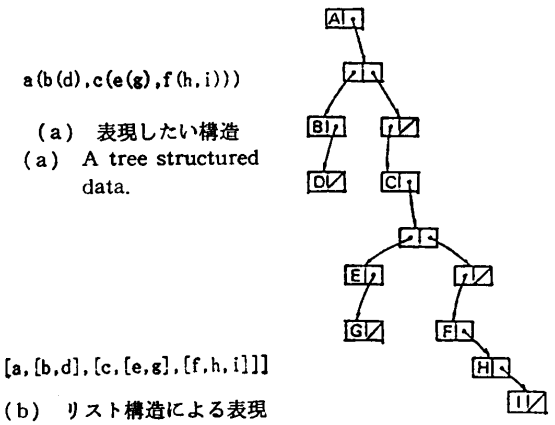

図 2 線形アドレス空間上での木構造の表現 (2)
Fig. 2 Representation of tree structured data (2).

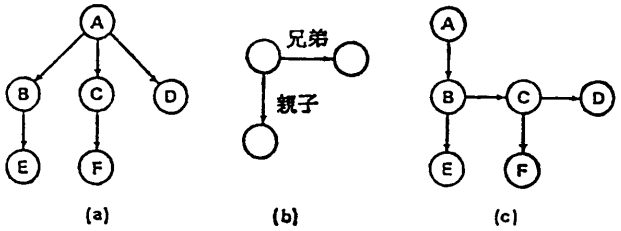

図 3 任意の木の二進木への変換
Fig. 3 Transformation from arbitrary to binary tree.

変換規則を設けることにより，二進木に変換できるので，この制約を設けても何ら一般性は失われない．例えば，図3の(a)は二進木でない木であるが，文献1)中にある規則，すなわち，右枝が兄弟関係を表現し，左枝が親子関係を表現するという規則を用いると（図3の(b)），この木は図3の(c)のような二進木になる．lisp で扱われる二進木は，リーフ以外のノードにはデータは記録されていないが，BTM では，すべてのノードにデータが記録されている二進木を扱うものとする．二進木の任意のノードにアクセスを行うために，BTM は，次の二つの機能を持っているものとする．

機能① ルートへの連想アクセス：

木に付けられた名前のみによって，直接，その木のルートをアクセスすることができる．

機能② ノード間の順序付きアクセス：

現在注目しているノードから，その親ノードと子ノードへ直接アクセスできる．

木構造を扱う多くのアプリケーションは木のルートから順に構造をたぐるので，これらの機能を持たせることにより，BTM をそれらのアプリケーションに応用することができる．

2.3 BTM の実現

上に定義されたような機能を持つ BTM の実現については，本論文では，既存の線形アドレス空間を持つメモリに，ルートへの連想アクセス機能，ノード間の順序付きアクセス機能を実現する制御回路を加える，という方法について検討する．

2.3.1 BTM 関数

ルートへの連想アクセス機能は木の名前からそれぞれの木のルートのアドレスを検索するハッシュ表を作る，または連想メモリを用いれば実現することができる．ノード間の順序付きアクセス機能に関しては，既存のメモリのセルは，2個の隣接セルしか持っていないため，二進木の3個の隣接ノード，親ノードと2個の子ノードを物理的につなぐことはできない．したがって，我々の目的のためには，論理的には接続されて

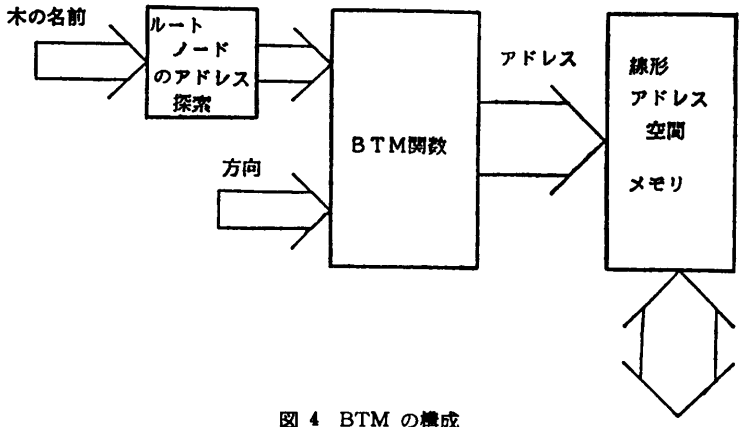

図 4 BTM の構成
Fig. 4 Structure of BTM.

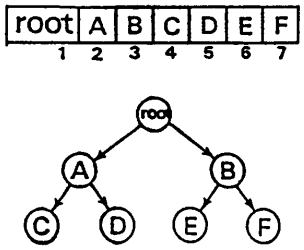

図 5 vector 関数による二進木のレイアウト
Fig. 5 A binary tree stored in *vector* function.

いるが，物理的に隣接していないノード間の相互関係を，ポインタを用いずに示すという問題を解決する必要がある．これには，メモリのアドレス空間を定義域とする，F (Node, Dir) というアドレス計算を行う関数を設計するという手法で対処する．ここで，引数 Node は現在注目しているノードのアドレスであり，引数 Dir は次に注目するノードの方向，すなわち，Dir＝{上，左，右} である．この関数を BTM 関数と呼ぶ．

BTM 関数は次の三つの条件を満足しなければならない．

① 一価関数である

② $\mathrm{Node}=F(F(\mathrm{Node}, \text{左}), \text{上})$
$=F(F(\mathrm{Node}, \text{右}), \text{上})$

③ メモリ空間が十分に利用できる

このような関数を用いて，二進木をメモリにレイアウトする．つまり，この関数に現在注目しているノードのアドレスを与えることにより，あるノードの親ノード，あるいは子ノードのアドレスが，この関数の計算結果として求められるようにする．それゆえ，ノード間はポインタではなく，関数により連結されていることになる．以上をまとめると，BTM は図 4 のような構成をとることになる．

我々は BTM 関数として採用する関数の候補として以下の三種類の関数について検討した．

① vector 関数

これは文献 2) に示されている完全二進木の表現法である．

$$F(\mathrm{Node}, \mathrm{Dir})=\begin{cases}-2\times\mathrm{Node} & \mathrm{Dir}=\text{左}\\ -2\times\mathrm{Node}+1 & \mathrm{Dir}=\text{右}\quad(1)\\ -\mathrm{Node}/2 & \mathrm{Dir}=\text{上}\end{cases}$$

この関数によって記録された二進木の例を図 5 に示す．このような関数は完全二進木のみに適する．任意の二進木に対してこの関数を適用すると，メモリの使用効率が非常に悪くなる．

② step count 関数

vector 関数を拡張して任意の二進木に対するメモリ使用効率を改善することを考える．関数 F に State という変数と Step という補助変数を導入して，

F (Node, Dir, State, Step)

とする．

State は 4 ビットのタグで，木構造中のあるノードの周りの構造（子ノードの数，兄弟ノードの有無及び親ノードから見た方向など）を表現する．左から第 1 番目のビット (brother bit) を 0 にすると兄弟ノードがないことを，1 にすると兄弟ノードがあることを示す．第 2 番目のビット (position bit) は，0 であると自分が親ノードの左のノードであることを，1 であると右のノードであることを示す．残った 2 ビット (son bit) により子ノードの数 (0, 1, 2) を示す．また，この 2 bit を双方 1 にすることにより，ルートノードであることを示す．この場合，position bit と brother bit は，元の意味が失われ，子ノードの数を示すこととする．Step という補助変数は木の二分岐の情報を記録する．初期値は 0 で，1 回二分岐したら（すなわち，子ノードが二つあれば，）インクリメントされる．子ノードが一つの場合には，Step は変化しない．

次に関数の定義である．ここで，定義中の変数 State の位置にある◆は，0 あるいは 1 という意味である．

$F(\mathrm{Node}, \text{左}, ◆◆\ 10, \mathrm{Step})=\mathrm{Node}+2^{\mathrm{Step}}$
計算後，Step をインクリメント
$F(\mathrm{Node}, \text{右}, ◆◆\ 10, \mathrm{Step})=\mathrm{Node}+2^{(\mathrm{Step}+1)}$

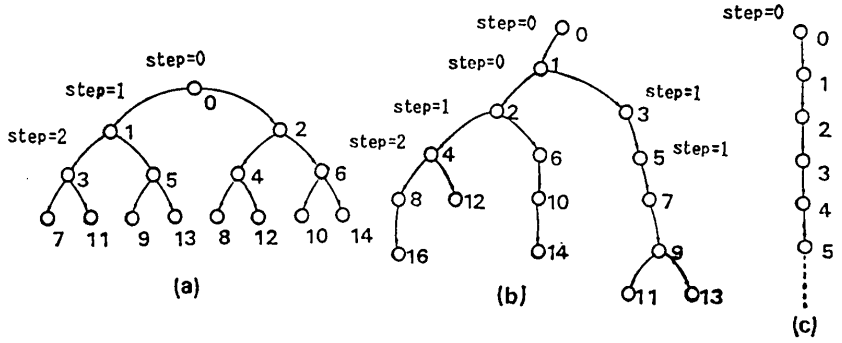

図 6 step count 関数による二進木のレイアウト
Fig. 6 Binary trees stored in *step count* function.

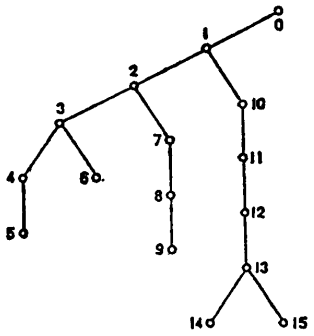

図 7 差値関数による二進木のレイアウト
Fig. 7 A binary trees stored in *Distance* function.

計算後，Step をインクリメント

F(Node, 左 or 右, ◆◆ 01, Step)=Node+2^{Step}

F(Node, 上, 11 ◆◆, Step)=Node−2^{Step}

計算後，Step をインクリメント

F(Node, 上, 10 ◆◆, Step)=Node−$2^{(Step-1)}$

計算後，Step をインクリメント

F(Node, 上, 0 ◆◆◆, Step)=Node−2^{Step} (2)

この関数を利用した二進木のレイアウトの例を図 6 に示す．図 6 (a) は完全二進木である．(b) は一般の二進木の例である．(c) は二進木の一つの特殊な例，リストである．この場合には二分岐がないので，補助変数 Step の値は変化しない．

③ 差値関数

関数②の欠点として，枝の深さがバランスしていない場合には，メモリ利用効率が悪化することがある．そこで，次の関数を考える．

$$F(\text{Node, Dif, Dir})=\begin{cases}\text{Node}+1 & \text{Dir}=\text{左}\\ \text{Node}+\text{Dif} & \text{Dir}=\text{右}\end{cases} \quad (3)$$

Dif というのはこのノードの左側の部分木のノード数に 1 を加えたものである．この方式によるレイアウトの例を図 7 に示す．

この三種類の関数を比較してみる．関数① (vector 表現法) は，完全二進木以外の木に対して適用すると，木を格納しているメモリの連続領域中にノードを表現していないセルが多量に現れるため，メモリ利用率が悪いという問題がある．関数の形を検討すれば明らかであるが，木のノード数を n とすると，最悪，$O(2^n)$ ものメモリ領域が必要である．しかし，どのリーフに対しても容易に 2 個の子ノードを接続することができ，どの二分岐していないノードに対してもあと 1 個の子ノードを接続することができるので，動的に変化する構造に対する対応性は高い．また関数の実現も容易である．利用率がある程度高くなることが保証できれば，動的な木構造データの処理に関して，非常に効率的な方法であると考えられる．

関数② (step count 関数) は，親ノードや子ノードと自分の間の相対位置に関する情報を各ノードに付加することにより，自分と親ノード，自分と子ノードの間でそれぞれ二分岐しない場合のメモリ利用効率を関数①に比較して改善している．関数②では，各ノードに対して，ノードのデータを表現する以外に余分に増加する情報量は 4 ビット (変数 State 用) である．また，補助変数 Step を保持しておくために，1 個のレジスタを設置する必要がある．関数②が有効な適用対象は，枝の深さのバランスが取れている動的な木である．枝の深さのバランスが取れていない場合，やはり関数の形を検討すれば明らかであるが，木のノードの数を n とすると，最悪，$O(2^n)$ ものメモリ領域が必要である．どのリーフに対しても容易に 2 個の子ノードを接続することはできるが，二分岐していないノードに対してあと 1 個の子ノードを接続することが難しい．つまり，構造をいったん決めると，決められた構造を変化させることは難しい．

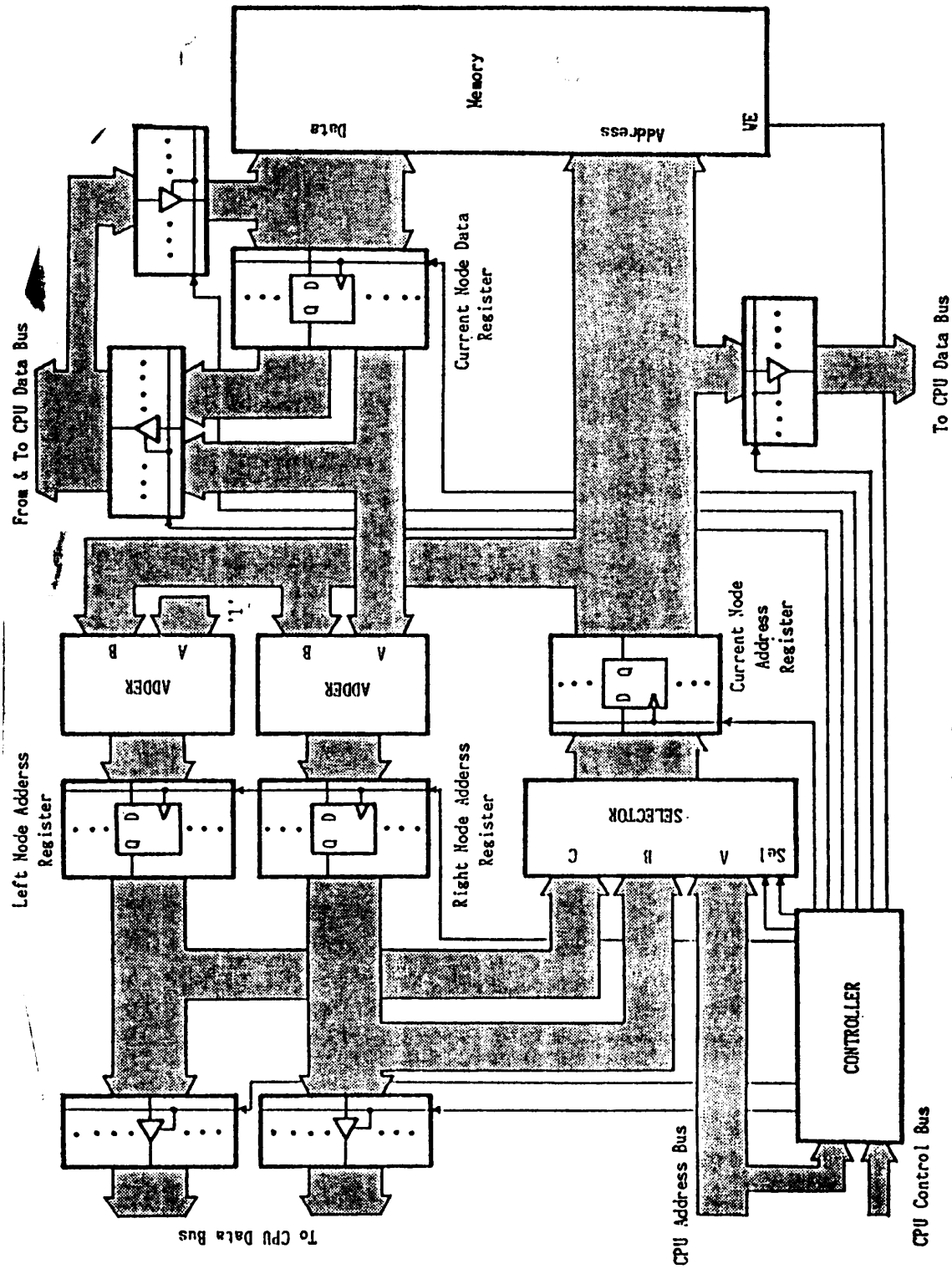

図 8 BTM 関数のハードウェアによる実現例
Fig. 8 An implementation example of BTM.

関数③（差値関数）は，あるノードの左側の部分木の大きさを前もって決めておくことにより，木を格納しているメモリの連続領域中にノードを表現していないセルを出現させないので，関数①，②のようなメモリ空間の浪費という問題はない．しかし，このために，この関数は静的な木構造データ，すなわち，構造がすべて決定された木についてしか適用することができない．さらに，右の子ノードから親ノードへの，注目するノードの変更ができない．関数③においては，各ノードに対して，ノードのデータを表現する以外に，余分に増える情報量は log (Difmax) ビット（底は2，Difmax は式(3)における Dif の最大値）である．

2.3.2 BTM の構成例

BTM 方式は，木構造の表現に今まで必要であったポインタをなくすことによって，ポインタに対するアクセスを削減する．ここで，ノード間の順序付きアクセス機能を実現するのに必要な，BTM 関数を高速に計算することのできるハードウェアの存在を仮定すれば，メモリに対するアクセスを削減することにより，Prolog 処理系の基本演算である木構造データ処理の効率の向上が期待できる．

図8のようなハードウェアを持てば BTM 関数は十分高速に計算できる．このハードウェアは，前にアクセスされたノードのデータから関数③（差値関数）を用いて左側の子ノードと右側の子ノードの双方のアドレスを並列に計算して，その結果をレジスタに格納しておき，次のアクセスの際にその結果を使用するものである．これは，CPU 自身の状態には関係なく実行することができるので，アドレス計算と命令の実行を並列に行うことが可能である．また，ルートへの連想アクセス機能を受け持つハッシュまたは連想メモリから得られるルートノードのアドレスを用いて CPU がルートノードに対してアクセスできるように，線形アドレス空間を持つメモリとしても動作する．さらに二分岐している木を扱うために，現在注目しているノード，左側の子ノード，右側の子ノードのそれぞれのアドレスを CPU から直接読み出せるようにもなっている．

関数①（vector 関数）や関数②（step count 関数）の BTM 関数ハードウェアもこれと同様にして設計することができる．これらの関数の実現のためにはシフタが必要であるが，バレル・シフタを用いると計算に必要な時間を短縮できる．

以上に述べてきたような，既存のメモリに対してアドレス関数計算の専用ハードウェアを付加して，BTM を実現するというアプローチの問題点としては，線形アドレス空間を基本にするので，動的なデータ構造の扱いやすさとメモリの使用率，この二つの要求を同時に満足させる関数の設計をすることが難しいという点が挙げられる．しかし，各処理対象の特徴または要求によって，適当な関数を選択することにより，この問題に対処することができる．例えば，Prolog 処理系に関して言えば，ヒープ領域のデータ，例えば，プログラムは静的なデータであり，実行環境は動的なデータであるので，それぞれに適当な方式を採用することができる．

3. BTM に基づいた Prolog マシン

BTM の有効性の評価の第一歩として，我々は，BTM を Prolog のプログラム領域のデータ表現に使用することにより得られる効果について調べた．この節では，この評価を得るために実装した Prolog システムについて述べる．

3.1 Prolog における木構造データの二進木表現

① 構造データの表現

述語の引数として現れる fn (arg 1, arg 2, ⋯, arg N) のような構造データを二進木で表現するために，**図9**に示すように，右側の枝で関数記号と引数の間の関係を表現し，左側の枝で各引数の間の関係を表現することにする．各引数がさらに構造データである場合も，同様にこの方法で表現する．例えば，fn (fn 1 (X 1, Y 1), fn 2 (X 2, Y 2), Z) という構造データならば，**図10**(1)のような二進木になる．リストは，ノード数を減少させるために，本来の dot functor を用いた構造で表現するのではなく，図10(2)に示すように，各要素を左側の枝で結合した形で表現する．これはリストの CDR-cording 方式[3]の二進木での表現である．さらにノード数を減少させるために，“|” という中置関数記号を省略することにした．このため，[X|Y] と [X, Y] のようなデータを区別するときは，cdr 部

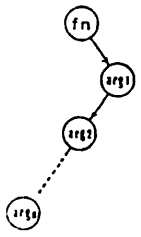

図9 二進木による Prolog の構造データの表現規則
Fig. 9 BTM expressions of fn (arg 1, arg 2, ⋯, arg N).

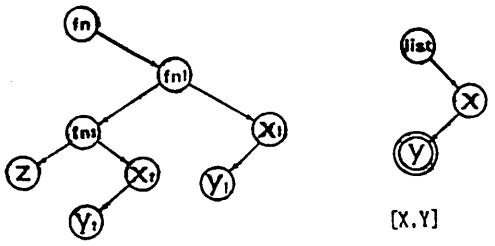

図 10(1) 二進木による fn(fn 1(X 1, Y 1), fn 2(X 2, Y 2), Z) の表現
Fig. 10(1) BTM expressions of fn (fn 1(X 1, Y 1), fn 2(X 2, Y 2), Z).

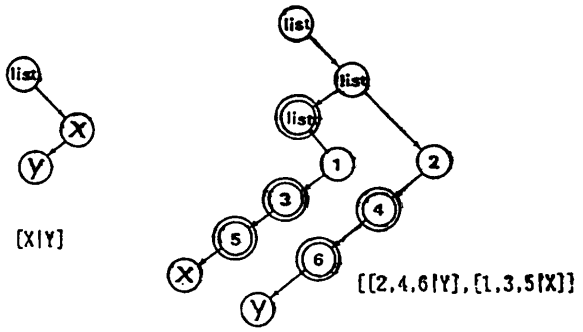

図 10(2) 二進木によるリストの表現
Fig. 10(2) BTM expressions of list.

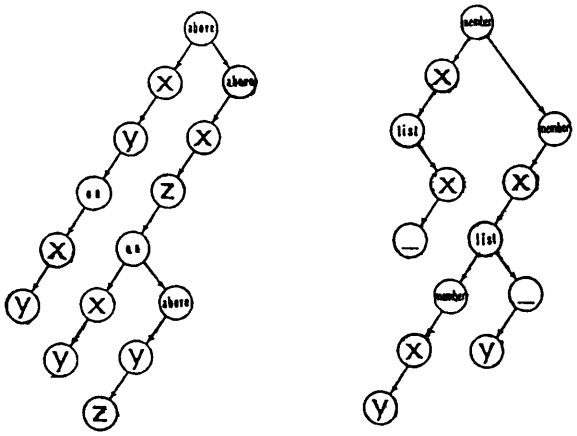

図 10(3) 二進木による節の表現
Fig. 10(3) BTM expressions of clause.

がさらにリストであることを示すために，リスト・タグを付ける．図10(2)中の二重丸の付いたノードはリスト・タグの付いているノードである．

② プログラムの表現

プログラム中で，述語記号が同じで，その arity も同じである head を持つ節の集合を一つの木にしたものをプログラム木と呼ぶ．このプログラム木を前に述べた方法を用いて二進木で表現する．**図12**に示す member と above に例をとれば，それらは図10(3)のような二進木により表現される．各節の OR 関係は head の述語記号を表すノードの右側の枝を用いて表現されている．

従来の Prolog のデータ表現方式と比較するため，最も代表的な Prolog 処理系，DEC-10 Prolog[4] のデータ表現方式を**図11**(1)-(3)に列挙する．DEC-10 Prolog では述語記号や関数記号とそれらの引数の間の関係をメモリの線形空間で表現して，各引数のとる値が構造を持つ場合にポインタを用いてその値を指す．これに対して，BTM 方式では構造体へのポインタを全部省略している．

3.2 BTM 関数の選択とデータ・フォーマットの設定

ここでは，我々は BTM をプログラム木を格納するプログラム領域に対して適用することを考える．節の assert と retract が起きない限り，プログラム木は構造が全く変化しないので，BTM 関数としては差値関数を採用した．1ワードは3ビットの Tag 部，5ビットの Dif 部，さらに，データ部から成る（図

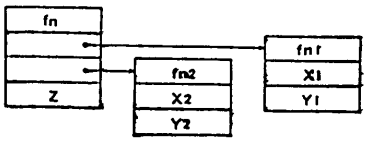

図 11(1) DEC-10 Prolog における fn(fn 1(X 1, Y 1), fn 2(X 2, Y 2), Z) の表現
Fig. 11(1) DEC-10 Prolog expressions of fn(fn 1(X 1, Y 1), fn 2(X 2, Y 2), Z).

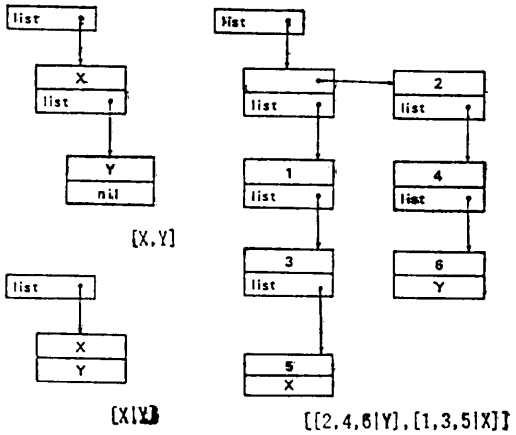

図 11(2) DEC-10 Prolog におけるリストの表現
Fig. 11(2) DEC-10 Prolog expressions of list.

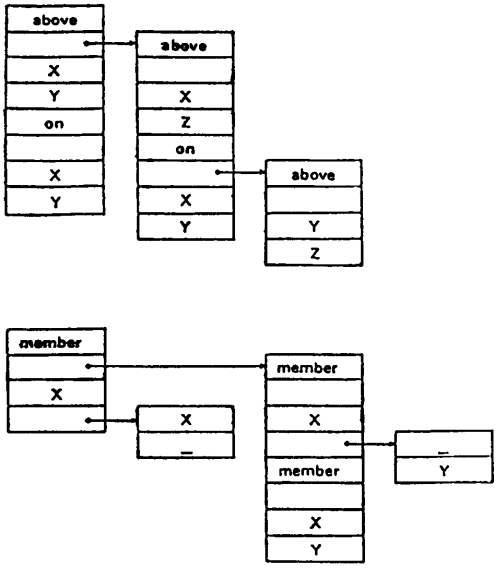

図 11(3) DEC-10 Prolog における節の表現
Fig. 11(3) DEC-10 Prolog expressions of clause.

```
above( X,Y ):- on( X,Y ).
above( X,Z ):- on( X,Y ), above( Y,Z ).

member( X,[X|_] ).
member( X,[_|Y] ):- member( X,Y ).
```

図 12 above と member のプログラム
Fig. 12 Definitions of *above* and *member*.

13). システムを VAX-11/750 上の，UNIX4. 2BSD 上に，言語Cでインプリメントした関係上，1ワードは32ビットであるので，データ部は24ビットとなる．

タグは次の8種類である．

- Atom アトムと関数記号
- LAtom リスト・タグが付いているアトム
- Int 整数
- LInt リスト・タグが付いている整数
- Var 変数
- LVar リスト・タグが付いている変数
- Ref 共有構造へのポインタ（環境スタックのみに使う）
- Point 共有変数へのポインタ（環境スタックのみに使う）

ここで，Ref と Point の二つのタグは，共有データの表現のためにプログラムの実行中に使用され，プログラム領域では使用されていない．

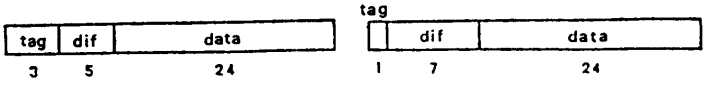

図 13 各ノードのデータフォーマット　拡張されたデータフォーマット
Fig. 13 Data format. Extended data format.

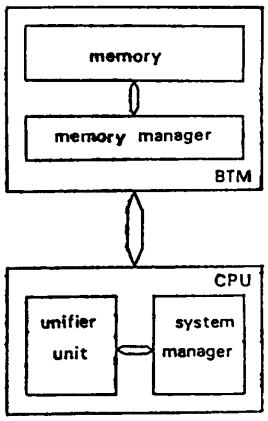

図 14 シミュレータのシステム構成
Fig. 14 System organization.

また，Dif が5ビットのみであるので，右側の部分木のサイズは最大31ノードしか許されないように思われるが，Dif を拡張することにより，右側の部分木の最大ノード数を127まで増加させることができる．Dif≠0 の時には，使われるタグは Atom と LAtom の二つであるので，タグの余った2ビットを利用して，Dif の長さを7ビットまで拡大することができる．

3.3 システム構成と実行メカニズム

システムの構成を図14に示す．システムは CPU 部と BTM 部から構成される．BTM 部はメモリとメモリ管理部から構成される．メモリ管理部は CPU 側から図15に示す BTM へのアクセスについての命令を受けとり，これらの命令に従って，メモリアクセスを行う部分である．CPU 部分はユニファイヤ部とシステム制御部に分かれる．ユニファイヤ部はタグの解析，構造の決定，変数のバインディングなどを行いながらユニフィケーションを行う．システム制御部は導出の制御，I/O 処理などの役割を受け持つ．Prolog における環境の表現は構造共有と構造コピーの二つの方式に分かれる[5]．本システムでは，構造共有方式を採用したが，構造コピー方式についても検討する予定である．環境の管理には，DEC-10 Prolog と同様，三つのスタック—global, local, trail を使用している．

4. 評価結果

前章で述べた BTM に基づいた Prolog システムにおける BTM の有効性に関して，次のような評価を行った．

```
Rrig(Di,Ai)          Read Ai's right, and assign to Di.
Rlef(Di,Ai)          Read Ai's left, and assign to Di.
Rfat(Di,Ai)          Read Ai's father, and assign to Di.
RrPl(Di,Ai,STAi)     Read Ai's right, and assign to Di.
                     Push Ai's left into STAi.
RlPr(Di,Ai,STAi)     Read Ai's left, and assign to Di.
                     Push Ai's right into STAi.
Rsta(Di,STAi)        Pop  STAi from Ai. Read Ai, and assign to Di.
Rroo(Di,Ai)          Read a tree root named Di,
                     assign root's address to Ai.
Rdir(Di,Ai)          Read Ai, assign to Di.
-------------------------------------------------------------------
Wrig(Di,Ai)          Write Di into Ai's right.
Wlef(Di,Ai)          Write Di into Ai's left.
WrPc(Di,Ai,STAi)     Write Di into Ai's right,
                     and push Ai into STAi.
WlPc(Di,Ai,STAi)     Write Di into Ai's left,
                     and push Ai into STAi.
WstaR(Di,Ai,STAi)    Pop  STAi from Ai, and write Di into Ai's right
Wroo(Di,Ai)          write a tree root named Di,
                     assign root's address to Ai.
Wdir(Di,Ai)          Write Di into Di.
-------------------------------------------------------------------
SetStaR(Ai,STAi)     Push Ai's right into STAi.
SetStaL(Ai,STAi)     Push Ai's left into STAi.
SetSta(Ai,STAi)      Push Ai into STAi.
-------------------------------------------------------------------

      Here, i = 0 or 1.
      Rigister:  D1 and D2  ( for keep data )
      Rigister:  A1 and A2  ( for keep address )
      Stack:     STA1 and STA2
```

図 15 BTM への命令
Fig. 15 The instructions of BTM.

1. BTM 方式を利用すると，メモリ要求量をどの程度削減することができるか．
2. BTM 方式を利用すると，メモリのアクセス回数をどの程度減少することができるか．

BTM の研究開発の目標は2章に述べたように，非数値処理における主な処理対象としての木構造データを効率よく処理することである．2.1 節の図1の例について考えると，BTM はアクセス量とメモリの使用量を約 40% 減らすことが期待できるが，この例は簡単な例であるので，実際の様々な応用プログラムについての一般的な目安にはならない．したがって，本研究では，文献 6) に発表されたベンチマークといくつかの応用実例[4),7),8)]を取り挙げて，評価の対象とした．**表1**は評価プログラムの一覧表である．**表2**と**表3**のプログラム番号は表1で付けられているプログラムの番号と一致している．また，比較対象を DEC-10 Prolog とした．

評価の方法は非常に簡単である．メモリ要求量の比較は次のように行う．プログラムを consult する時の BTM への全書き込み回数 (BW と呼ぶ) をカウントする．BW が BTM でのメモリ要求量となる．さらに，consult したプログラム木のノードのうち，関数記号を表現するノードの個数 (FW と呼ぶ) もカウントする．DEC-10 Prolog で採用された方法では，構造を表現するために，一つの関数記号に対して必ず一つのポインタが必要であるので，FW はポインタの数に等しい．したがって，DEC-10 Prolog 方式を採用した際のメモリ要求量 (DW と呼ぶ) は BW＋FW ワードになる．リスト構造は我々のシステムでは各要素を左側の枝で結合した形で表現されるが，DEC-10 Prolog 方式では dot functor を用いた構造で表現される．我々のシステムを用いて DEC-10 Prolog 方式におけるメモリ要求量やアクセス量を調べるためには，プログラム中で [A|B] と書かれているリスト構造を，cons (X, Y) と書き換えればよい．各評価プログラムを consult した後に得られた結果を表2に示す．この表によると，平均的に BTM 方式は，DEC-10 Prolog 方式より20～48% のメモリ領域を削減することができる．

アクセス量の比較に関しては，前述の方法と同様な方法で評価をした．プログラムの実行中に発生した BTM への全アクセス回数 (BA と呼ぶ) と関数記号を表現するノードへのアクセス回数 (FA と呼ぶ) をそれぞれカウントする．DEC-10 Prolog 方式の場合は，各関数記号に対応しているポインタへのアクセスも必要になるので，この場合の全アクセス回数 (DA と呼ぶ) は BA＋FA となる．カウントした結果を表3に示す．これは構造共有方式を採用した場合のアクセス量の結果である．これらの結果からアクセス量に関して，次の結論が得られる．

アトム，変数間のユニフィケーション	18～20% 減少
データベース検索	17～33% 減少
リスト処理	22～24% 減少
応用プログラム	19～36% 減少

表1で説明しているように，評価プログラム1の中のプログラム，Atom-1, Atom-5, Var-1, Var-5 は，単にアトム，変数間のユニフィケーションを行うだけで，木構造の引数は存在しない．しかしながら，この

表 1 評価プログラム
Table 1 Illustration of benchmark.

No.	program name	Ref.	illustrate
1	Atom-1 Atom-5	6	Unification of Atom, Arity of one arity of five. For 100 iterations.
	Var-1 Var-5	6	Unification of variables. Arity of one and arity of five.
	Con-1 Con-5	6	Unification of constant structure. Arity one and arity five. For 100 iterations.
	Str-1 Str-5	6	Unification of nonconstant structure. Arity one and arity five. For 100 iterations.
	Str-Var-1 Str-Var-5	6	Unification of variables with structure. Arity one and arity five. For 100 iterations.
	Var-Str-1 Var-Str-5	6	Unification of structures with variable. Arity one and arity five. For 100 iterations.
	Det-Call Ndet-Call Shallow-Back Deep-Back	6	Deterministic simple call Nondeterministic simple call Sallow backtracking Deep backtracking
2	Key-First First Key-Last Last Key-Middle Middle	6	Database search. Get first clause with primary key, get first clause, get last clause with primary key, get last clause, get middle clause with primary key, and get middle clause. For all of them 100 iterations.
3	Rev-30	6	List30 naive reverse, for 100 iterations
4	Sort-50	6	List50 quick sort, for 100 iterations
5	Srev-4 Srev-5	6	Slow reverse. 4 elements. 5 elements.
6	8-Queen-1 8-Queen-all	6	Eight queen for one solution. Eight queen for all solution.
7	Deriv	4	Symbolic differentiation
8	mis	8	Missionaries and cannibals.
9	parser	7	A simple parser of English.

表 2 メモリ要求量の比較
Table 2 Comparison of the memory space.

No.	spaces in word DEC_10 Prolog (DW)	spaces in word BTM (BW)	ratios (DW-BW)/DW
1	1335	1046	22%
2	1233	984	20%
3	279	159	43%
4	413	216	48%
5	326	233	28%
6	386	308	20%
7	460	355	22%
8	718	555	23%
9	594	453	24%

表 3 アクセス回数の比較
Table 3 Comparison of the access time.

No.	program name	DEC-10 Prolog's access times BA	BTM's total access times DA	ratios R = (DA-BA)/DA
1	Atom-1	5928	4720	20%
	Atom-5	6737	5529	18%
	Var-1	5928	4720	20%
	Var-5	6737	5529	18%
	Con-1	6537	5129	22%
	Con-5	9737	7529	23%
	Str-1	6537	5129	22%
	Str-5	9737	7529	23%
	Str-Var-1	6037	4729	22%
	Str-Var-5	8831	7129	19%
	Var-Str-1	6337	5029	21%
	Var-Str-5	8837	7129	19%
	Det-Call	6837	5429	21%
	Ndet-Call	5937	4729	20%
	Shallow-Back	11928	9720	18%
	Deep-Back	25537	21029	18%
2	Key-First	6137	4929	20%
	First	6137	4929	20%
	Key-Last	57429	38721	33%
	Last	111137	92429	17%
	Key-Middle	49337	37329	24%
	Middle	70937	58929	17%
3	Rev-30	10864	8273	24%
4	Sort-50	12303	9350	24%
5	Srev-4	3984	3080	23%
	Srev-5	15679	12165	22%
6	8-Queen-1	127091	102374	19%
	8-Queen-all	2053698	1653499	19%
7	Diff	4665	3086	34%
8	mis	55508	39230	29%
9	parser	543	347	36%

ようなプログラムについても，18～20%程度アクセス回数を減少させることができる．なぜならば，プログラム木へのアクセス時にポインタへのアクセスを行う必要がないからである．評価プログラム2はデータ・ベース検索の例で，これも木構造を持つ引数のないプログラムである．その中では，第一引数をキーワードとしたデータ・ベースの全探索の例が関数記号へのアクセス回数の割合が一番大きく，約33%アクセス回数を減らすことができる．プログラム7は，この中で最も木構造を使用している記号微分の例である．このような例に対しては，約34%アクセス回数を減少させることができる．プログラム8と9はリスト構造と普通の木構造を両方使う例であり，その中では，英文の構文解析プログラムが約36%と，最も高い減少率が得られる．

まとめると，差値関数を用いたBTMを使うと，プログラム領域を20～48%削減することができ，プログラム領域へのアクセス回数を17～36%減少させることができる．ここで，前に述べたように，BTM関数を高速に計算することのできるハードウェアの存在を仮定すれば，メモリに対するアクセスを削減することにより，Prolog処理系の基本演算である木構造データ処理の効率が向上すると見られる．

基本機能のスピード・アップという効果を考えると，さまざまなProlog実行モデルに対してBTM方式を用いると，それぞれをより速く実行することができることが期待できる．特に，プログラム領域が多数のプロセッサに共用される並列処理モデルでは，メモリアクセスが競合するために，アクセス回数が減少することは非常に重要であり，また，データのコピーを頻繁に行う他の並列モデルにおいては，データのコンパクト性も非常に重要であり，この点でもBTMは有利である．

5. むすび

本論文では，BTMを提案し，その有効性を評価した．

評価を取るために実装したシステムは，アドレス計算関数として関数③を利用しているが，他の関数を使用しても，そのアクセス回数減少の効果は変わらない．ただし，メモリの利用効率が悪くなるという問題がある．2章で述べた関数③以外のBTM関数は，動的に変化する構造体の処理を行う場合に必要となる．本論文では，Prologのプログラムに対してBTMを適用した場合しか評価を行っていないが，今後，OR並列における変数環境の管理にBTMを利用することについて検討していく予定である．OR並列を実現する時は，木構造で環境を管理することが必要となるが，2章で述べた関数①または関数②を用いると，効

率よく動的な環境木を管理できることが期待できる.

謝辞 本研究に際し，貴重なご助言とご指導を頂いた，慶応義塾大学理工学部教授 所 真理雄博士に心より感謝いたします．また，日頃討論いただく慶応義塾大学理工学部電気工学科相磯研究室，所研究室の宮崎 淳氏，天野 英晴氏，石川 裕氏，秋山 泰氏に感謝いたします.

参考文献

1) Knuth, D. Z.: *The Art of Computer Programming*, Vol. 1, Addison Wesley, Reading (1968).
2) Wulf, W. A., Shaw, M. and Hilfinger, P. N.: *Fundamental Structures of Computer Science*, Addison Wesley, Reading (1981).
3) Clark, D. W.: An Empirical Study of List Structure in LISP, *CACM*, Vol. 20, No. 2, pp. 78-87 (1977).
4) Warren, D. H. D.: IMPLEMENTING PROLOG—Compiling Predicate Logic Program, D. A. I. Research Report, No. 39 (1977).
5) Bruynooghe, M.: The Memory Management of Prolog Implementations, *Logic Programming*, pp. 83-98, Academic Press, New York (1982).
6) 奥及 博：第三回 LISP コンテスト及び第一回 PROLOG コンテストの課題案，記号処理，28-4 (1984. 6. 15).
7) Clocksin, W. F. and Mellish, C. S.: *Programming in Prolog*, Springer-Verlag, Berlin, Heidelberg, New York (1981).
8) 中島秀之：Prolog，産業図書，東京(1983).

(昭和60年9月3日受付)
(昭和61年5月15日採録)

楊 容（正会員）

1955年生．1978年北京大学計算機科学技術科卒業．1981年中国科学院大学院修士課程修了．現在，慶応義塾大学理工学部博士課程在学中．ロジック・プログラミング，コンピュータ・アーキテクチャ，人工知能に興味を持つ．ACM 学生会員.

正畑 康郎（学生会員）

昭和37年生．昭和60年慶応義塾大学理工学部電気工学科卒業．現在，同大学院修士課程在学中．主に記号処理向きアーキテクチャに興味を持つ．IEEE，ACM 各学生会員.

相磯 秀夫（正会員）

昭和7年生．昭和30年慶応義塾大学工学部電気工学科卒業．昭和32年同大学院修士課程修了．大阪大学工学部助手を経て，通産省工業技術院電気試験所(電子技術総合研究所)に勤務．トランジスタ計算機の研究開発に従事．その間昭和35年から1年半米国イリノイ大学計算機研究所に留学．昭和46年慶応義塾大学工学部電気工学科教授，現在に至る．工学博士．著書「計算機アーキテクチャ」(岩波講座)ほか．電子通信学会，IEEE，ACM 各会員.