

項関係における高速検索手法^{†*}横 田 治 夫^{††} 北 上 始^{††} 服 部 彰^{††}

本稿では、単一化 (Unification) を使って知識ベースの検索をする RBU (Retrieval By Unification) 演算の高速化の手段として、項関係に対するインデクスの実現方法について報告する。項関係とは、変数も取り扱うことが可能な構造体である項 (Term) を格納したテーブルである。また、関係代数演算に単一化を導入して項関係から適当な項を検索する演算を RBU 演算と呼ぶ。項関係上の RBU 演算により、大容量の知識ベースを対象とした各種の推論が行える。ここで提案するインデクスは、ハッシングとトライ (Trie) 構造と呼ばれる一種の木構造を組み合わせて RBU における比較処理ならびにバックトラック処理の回数を抑えるものである。トライ構造によるインデクスの効果を高めるため、項を LOSR のセル構造で表現し、FIFO を用いて LOSR の項どうしを単一化するアルゴリズムを示す。また、試作したプロトタイプの実験処理に要する時間を計測し、インデクスの検索処理における高速化の効果を確かめる。特に、各種形態の項関係に対して、トライ構造とハッシングの組み合わせが有効であることを示す。さらに、挿入処理に要する時間を計測し、更新処理におけるインデクスの維持のためのオーバーヘッドがわずかであることを示す。

1. はじめに

RBU (Retrieval By Unification) 演算は、第五世代コンピュータ・プロジェクトにおいて、知識ベースシステムの演算の一つとして提案された¹⁾。関係データベースにおける関係代数演算を知識検索用に拡張している。変数を含んだ構造体である項で知識を表現し、項の集合である項関係と呼ばれるテーブルから検索条件に単一化 (Unify) 可能な項を選択してくる。関係データベースが互いに関連し合った大量のデータに対して検索しやすい環境を与えたように、RBU もそれぞれに関係を持った大量の知識の塊をうまく取り扱うことを目指している。

RBU 演算による知識ベース処理の例の一つとして、RBU 演算の繰り返しによる SLD 演繹の実現がある¹⁾。SLD 演繹は、一階述語論理の一部であるホーン節論理の演繹アルゴリズムの一つで、Prolog の実行メカニズムの基にもなっている²⁾。このため、ホーン節で意味ネットワーク的な知識を表現する DCKR³⁾ などを利用することにより、RBU 演算の組み合わせで階層的な知識を検索することができる。また、並列論理型言語の一つである GHC (Guarded Horn Clauses)⁴⁾ と RBU 演算を組み合わせて並列プロダクション・システムを実現する方法も提案されている⁵⁾。GHC はストリームを使った並列処理を記述す

るのには強力な言語であるが、知識ベースのようなグローバルなデータを扱うのは得意でない。

演繹処理やプロダクション・システムなどの知識ベースを利用する処理においては、検索時間が全体の処理効率に大きく影響してくる。項に対する高速検索の手段として、単一化エンジンを用いる方法^{1), 6), 7)} など専用ハードウェアを使った方法も提案されているが、ここではソフトウェアのみの実現を前提として、インデクスによる高速化について検討する。

Prolog システムでは、コンパイルコードのインデクスとしてハッシングを使っているものがある。また、単一化を使った結合演算のためにハッシュ・ベクタを使った方法の提案もされている⁸⁾。ハッシングは、ハッシュ・キーの対象となる内容が重ならないような場合に、大量の項目を検索するためのインデクスとして非常に強力である。しかし、ハッシュ・キーの内容が重なった場合には、ハッシュ・エントリの競合が生じインデクスの効果が薄らぐ。RBU では比較的似た構造の項に対して単一化可能なものを探すため、適当なハッシュ・キーを選ぶことが難しい。また、単一化を使った検索では、変数の束縛 (binding) をしなおすためのバックトラックが必要であり、単純なハッシングだけでは次に束縛すべき内容をすぐに見つけることができない。さらに、インデクスを設けることにより、更新処理においてインデクスを維持するための処理が必要になる。このインデクス維持のオーバーヘッドも考慮する必要がある。

本稿では、RBU 演算を高速化するために、ハッシングとトライ (Trie) 構造と呼ばれる一種の木構造を使ったインデクスを提案し、そのインデクスを使って

[†] An Accelerated Retrieval Method for Term Relations by HARUO YOKOTA, HAJIME KITAKAMI and AKIRA HATTORI (FUJITSU LIMITED).

^{††} 富士通(株)

* 本研究は第五世代コンピュータプロジェクトの一環として行われたものである。

試作したプロトタイプの評価を行う。まず第2章で、項関係とRBU演算に関する定義を行い、第3章でインデクスの実現方法について述べる。また、インデクスの検索ならびに更新における影響を見るために試作したプロトタイプの評価について第4章で報告する。

2. 項関係とRBU演算

項の定義は一階述語論理⁹⁾のそれと同じであるが、ここでは項関係を定義するため集合を使う。

【定義】 F_n を n 引数の関数記号の有限集合、 V を次の式を満足する変数の可算無限集合とする。

$$\forall n, F_n \cap V = \emptyset.$$

【定義】 次のような T を項集合 (Term Set) と呼び、その要素 t を項 (Term) と呼ぶ。

- i) もし $t \in F_0$ または $t \in V$ ならば $t \in T$
- ii) もし $t_1, \dots, t_n \in T$, $f \in F_n$ ($n \geq 1$) ならば $f(t_1, \dots, t_n) \in T$

【定義】 $T_1, T_2, \dots, T_m$ を項集合としたとき、次のような TR^m を m 層性の項関係 (Term Relation) と呼ぶ。

$$T_1 \times T_2 \times \dots \times T_m \supset TR^m \quad (m \geq 1).$$

このとき次のような tt^m を項タプル (Term Tuple) と呼ぶ。

$$tt^m = (t_1, t_2, \dots, t_m) \in TR^m$$

項タプル tt^m の i 番目のアイテムを $tt^m[i]$ で表す。

六つの項タプルからなる2属性の項関係の例を表1に示す。本稿では、変数を大文字で始まる記号で表すことにする。表の先頭の項タプルの第1属性にある項 $p(X, g(Y))$ の中の変数 X が、項 $f(a, b)$ に束縛された場合、変数のスコープからこの項タプルの第2属性の項 $r(X, Y)$ は $r(f(a, b), Y)$ となる。このような変数束縛 $\theta = \{f(a, b)/X\}$ を代入 (Substitution) と呼ぶ。

項関係上の演算は、関係代数¹⁰⁾を基にして、単一化を導入して拡張している。関係代数には、集合和

表1 項関係の例

Table 1 Example of term relation.

第1属性	第2属性
$p(X, g(Y))$	$r(X, Y)$
$q(f(a, X), g(X))$	$r(f(a, X), X)$
$p(X, g(b))$	$r(h(a, b), f(a))$
$q(f(X, Y), g(c))$	$s(X, g(Y, c))$
$p(f(a, b), h(X))$	$s(a, g(b, c))$
$p(f(a, X), h(X))$	$s(a, X)$

表2 単一化制約の結果

Table 2 Result of unification-restriction.

第1属性	第2属性
$p(f(A, c), g(Y))$	$r(f(A, c), Y)$
$p(f(A, c), g(b))$	$r(h(a, b), f(a))$
$p(f(a, c), h(c))$	$s(a, c)$

(Union), 制約 (Restriction), 射影 (Projection), 結合 (Join), などの演算があり、RBUのプロトタイプではそれらのほとんどを実現し、結合と制約について単一化による拡張を行った。また、更新系の演算や定義系の演算についても実現した。ここでは、インデクスに焦点を絞るために、単純な選択演算である制約について述べる。

【定義】 代入 θ は、 $t_1\theta = t_2\theta$ となるとき、またそのときに限って、項 t_1 と t_2 の単一化作用素 (Unifier) と呼ばれる。

【定義】 項 t_1 と t_2 の単一化作用素 σ は、それらの項の任意の単一化作用素 θ に対してある代入 λ が存在し、 $\theta = \sigma \cdot \lambda$ となるとき、またそのときに限ってそれらの項の最汎単一化作用素 (Most General Unifier) と呼ばれる。

【定義】 単一化制約 (Unification-Restriction) とは、ある項関係 TR_a^m とその i 番目の属性に対する検索条件の項 t から次のような新しい項関係 TR_b^m を作る演算である。

$$tt_b^m \in TR_b^m \Leftrightarrow \exists tt_a^m \in TR_a^m, \exists \sigma, tt_a^m[i]\sigma = t\sigma,$$

$$tt_b^m[k] = tt_a^m[k]\sigma \quad (1 \leq k \leq m)$$

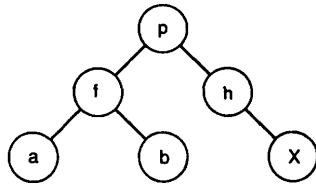
ここで、 σ は $tt_a^m[i]$ と t の最汎単一化作用素である。

表1の項関係に対して、第1属性が検索条件 $p(f(A, c), B)$ と単一化可能であるような項を検索してくる単一化制約を行った場合、1, 3, 6番目の項タプルが導かれ、表2に示すような項関係が生成される。ここで、それぞれの項タプルに対する最汎単一化作用素は、 $\{f(A, c)/X, g(Y)/B\}$, $\{f(A, c)/X, g(b)/B\}$, $\{c/X, a/A, h(c)/B\}$ である。

3. 知識検索システムの実現方式

3.1 項の表現方法

項を格納、検索するためには、なんらかの方法で項を表現する必要がある。項を木構造、単一化をその木の上のパターンマッチングと見なすことができる。つまり、変数は木中の対応する位置にある部分木が代入

図 1 項 $p(f(a, b), h(X))$ の木構造Fig. 1 Tree structure of term $p(f(a, b), h(X))$.

される。図 1 に項 $p(f(a, b), h(X))$ に対応する木構造を示す。この項が、 $p(Y, Z)$ と単一化されると、変数 Y に f を根とする部分木が、変数 Z に h を根とする部分木が代入される。

木構造の表現方法として、レベル順線形化表現 (LOSR: Level Order Sequential Representation) とファミリー順線形化表現 (FOSR: Family Order Sequential Representation) がある¹¹⁾。図 1 の木構造に対する LOSR は $[p-2, f-2, h-1, a-0, b-0, X]$ であり、FOSR は $[p-2, f-2, a-0, b-0, h-1, X]$ である。それぞれの表現の要素は、元の木構造を再現するために、その関数記号と引数の数 (つまり、そのノードにつながる枝の数) の組で表記する。

一般の Prolog のシステムは、FOSR を使っている。これは、FOSR では関数記号の引数とその関数記号に再帰的に続くため、繰り返し起こる Prolog の複雑な代入に向いているためである。一方、項の構造的な違いを判定するために要素を先頭から比較していく場合には、FOSR は LOSR に比べて多くの比較を必要とする。例えば図 1 の p の木が f と h の二つの部分木からなることは、LOSR だと f の部分木の大きさに関係なく 3 要素の比較で判定可能であるのに対し、FOSR では f の部分木全部を比較する必要がある。RBU が、単一化を何回も繰り返すことよりも、似た構造の多くの項の中から必要な項を早く探すことを目的としているため、RBU のプロトタイプでは LOSR を用いることにする。

項の長さが可変であるため、我々は LOSR の要素を格納するために図 2 のようなセルを使うことにする。1 番目と 2 番目のフィールドは、対応する要素の内容を表すためのものである。その要素が関数記号の場合には、第 1 フィールドに関数記号表 (Atom

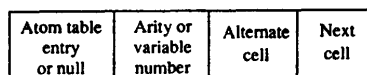


図 2 セル構造

Fig. 2 Cell structure

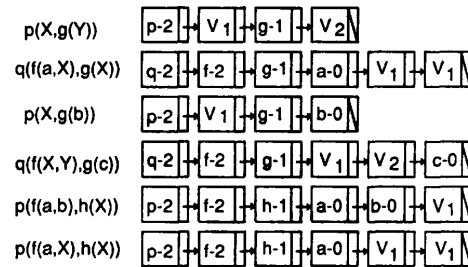


図 3 LOSR に対するセル構造

Fig. 3 Cell structure for LOSR of terms.

Table) の対応するエントリへのポインタが、第 2 フィールドにその引数の数が入られる。ここで、関数記号も可変長の文字ストリングであり、同一の関数記号が項関係中に何回も現れるため、直接セルに格納せずに別の関数記号表を用意している。要素が変数の場合には、第 1 フィールドには null ポインタを入れ、第 2 フィールドにその変数の番号を格納する。変数の記号は、単一化の間に付け替えられるため重要ではなく、変数のスコープに従って項タプル中で順番に番号付けを行う。この時、同一の変数には同一の番号を振る。3 番目のフィールドは後で述べるトライ構造を構成するためのもので、4 番目のフィールドには次のセルへのポインタを格納する。最後のセルで次のセルがない場合には、null ポインタを入れておく。

図 3 は、表 1 に示した項関係の第 1 属性に対する LOSR のセル構造である。ここでは、見やすさのため関数記号表へのポインタは省略して関数記号と引数の数の組で示し、変数は V と変数番号をしめす添字で表現している。また、第 3 フィールドは省略して、null ポインタは斜線で示している。

我々は、このセルを項を格納するのとインデックスを構成するために利用する。このため、図 3 に示したようなセル構造を単純セルリスト (Flat Cell List) と呼ぶことにする。

3.2 ハッシングとトライ構造

ハッシングは、高速検索のためによく用いられるインデックス手法の一つである¹²⁾。しかし、知識ベース処理のための項関係は多くの似た構造が格納されることが想定される⁵⁾ ため、ハッシングだけではその効果が期待できない。また RBU ではバックトラックを使って、単一化可能な項をすべて検索することを目的としているため、単一化可能な項をインデックスのなるべく近い位置に置くことが望ましい。そこで、我々はハッシングとトライ構造を組み合わせたインデックスを提案

する。

トライ (Trie) 構造とは、頭から見て同じ内容の要素をまとめた一種の木構造である¹²⁾。トライ構造は、一般には数値や文字の列を格納して、記憶容量を節約するために使われるが、ここでは項の LOSR の要素に適用してインデックスの一部として利用する。

LOSR の先頭の要素の等しい項を一つのトライ構造とし、頭から等しい部分は要素を共有する。このとき、前述したセルの第3フィールドを分岐に利用する。図3に示した項の集合をトライ構造に変換すると図4に示す $p-2$ と $q-2$ を根とする二つのトライ構造になる。例えば、 $p(X, g(Y))$ と $p(X, g(b))$ の LOSR の先頭の3要素が等しいため三つのセルを共有し、4番目のセルに分岐を持たせる。こうすることにより、似通った構造の項は、トライ構造上で近くに置かれることになる。

単一化制約のコストは、検索条件の項の要素と検索対象の項の要素の間の比較の回数に比例する。トライ構造上で、単一化を行うことにより比較の回数を大幅に減らすことができる。図3と図4の項の集合に対して、 $p(f(a, b), h(c))$ という検索条件で単一化制約を実行する場合を想定する。比較されるセルは、次に示すトレースのようになる。

【単純セルリストの場合 (図3)】

$p-2, V1 (=f-2), g-1 <fail>$
 $q-2 <fail>$
 $p-2, V1 (=f-2), g-1 <fail>$
 $q-2 <fail>$
 $p-2, f-2, h-1, a-0, b-0, V1 (=c-0) <success>$
 $p-2, f-2, h-1, a-0, V1 (=b-0), V1 <fail>$ ■

【トライ構造の場合 (図4)】

$p-2, V1 (=f-2), g-1 <fail>$
 $f-2, h-1, a-0, V1 (=b-0), V1 <fail>$
 $b-0, V1 (=c-0) <success>$

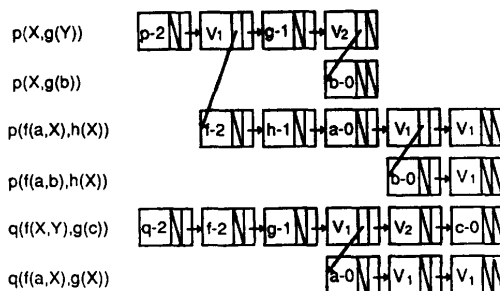


図4 項に対するトライ構造
 Fig. 4 Trie structures for terms.

$q-2 <fail>$ ■

このトレースから分かるように、単純セルリストだと最初の要素の比較が項の数分 (6回) 行われるのに、トライ構造を使うと先頭の要素の種類分 (2回) で済む。当然この違いは同じ要素を先頭に持つ項の比率によってさらに大きくなる。同じことは2番目以降の要素についても言える。一つの項の平均要素数を M 、項の数を N とすると、単純セルリストを使った場合の比較の回数は最悪で $M \times N$ になる。これは、検索対象のすべての項が、最後の要素を除いてすべて等しい場合と考えることができる。この場合、トライ構造を用いることにより $M+N$ に減らすことができる。

トライ構造は単に要素間の比較の回数を減らすだけでなく、バックトラック処理の回数も減らしていることに注意しなければならない。上のトレースにおいて、行の最後の $<fail>$ とか $<success>$ がバックトラックに対応する。この例では、6回から4回に減っている。トライ構造がうまく均衡が取れている場合には、バックトラックの数が大幅に減少する。比較の回数もこの効果によって $M+N$ からほとんど M 近くまで減らされる。上の例では、全体の比較の回数はトライ構造を使った場合と使わない場合で、20 から 11 に変化している。

我々は、似た構造の項と異なった構造の項のいずれの形態にも効果のあるインデックスを実現するために、トライ構造とハッシングを組み合わせることにした。LOSR の先頭の要素をハッシュ・キーとして、その要素を持つトライ構造へのポイントをハッシュ・テーブルに格納する (図5)。ハッシュ関数としては、関数記号表のエントリのビット列をその表の大きさに対応させて適当な長さに畳み込んで排他的論理和を

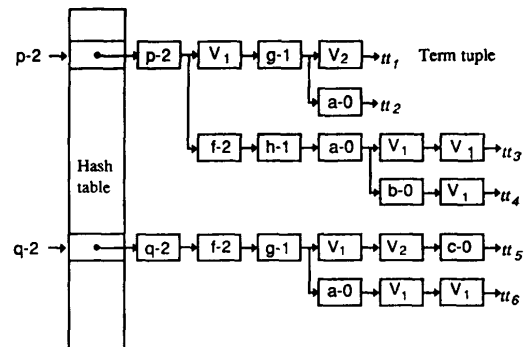


図5 ハッシングとトライ構造を組み合わせたインデックス
 Fig. 5 Combination of hashing and trie structures.

取ったものを使うことにした。こうすることにより、似た名前の関数記号が多い場合でも効率的にちらすことができる。

このようにハッシングと組み合わせたトライ構造は、見方を変えると、ハッシングの競合解消を効率的に行う機構と見ることもできる。トライ構造の葉に相当する部分のセルの次のセルを指すためのポインタは、対応する項をキー属性に持つ項タプルへのポインタとなる。基本的には、ハッシングは一つの検索条件に対して1回実行されるだけである。検索条件に単一化可能な項は、少なくとも先頭の要素が条件と同じはずであるから、そのエントリに接続されたトライ構造の中に含まれるはずである。つまり、ハッシングは探索空間を狭めるのに用いられ、その探索空間上で効率的に単一化を行うためにトライ構造を使うわけである。

3.3 LOSR 上の単一化

Prolog システム等で使われている単一化のアルゴリズムは、項が FOSR で表現されていることを前提としているため、RBU システムのためには LOSR 用の単一化アルゴリズムを用意する必要がある。

二つの項 $p(X, g(Y))$ と $p(f(a, b), g(c))$ は、変数 X に $f(a, b)$ が代入可能であり、変数 Y に c が代入可能であるため、単一化可能である。ここで、それぞれの項の LOSR は、 $[p-2, V_1, g-1, V_2]$ と $[p-2, f-2, g-1, a-0, b-0, c-0]$ となる。LOSR どうして単一化を行うためには、1 番目の変数 $V_1 (=X)$ には、単に対応する位置にある $f-2$ だけでなく、その引数である $[a-0, b-0]$ も代入される必要がある。また、2 番目の変数 $V_2 (=Y)$ には、そのままでは対応する位置にない要素 $c-0$ を代入しなければならない。これらのためには、 $[p-2, V_1, g-1, V_1', V_2]$ という仮想的な LOSR を作る必要がある。我々は、FIFO (First-In-First-Out list) を使ってこの変数拡張を行うことにする。つまり、単一化の時に、拡張すべき変数かそれとも拡張すべきでない要素かを示すためのフラグを、その要素の引数の分だけ FIFO に入れていくようにする。以下にその FIFO を用いた単一化アルゴリズムを示す。

[LOSR 上の単一化アルゴリズム]

Step 1: L_A と L_B を単一化の対象の項の LOSR とする。初期設定として、非拡張フラグ n を二つの FIFO W_A と W_B の先頭に入れる。

Step 2: if W_A と W_B の両方が空 then 単一化成功

else W_A と W_B の先頭からフラグ A と B を取る。

Step 3: case A も B も n の場合

L_A と L_B の先頭から要素 E_A と E_B を取る

if $E_A = F_A \cdot K_A$ かつ $E_B = F_B \cdot K_B$ then

if $F_A = F_B$ かつ $K_A = K_B$ then

K 個の n を W_A と W_B に入れ、Step 2 へ

else 単一化失敗

else if $E_{A/B} = F \cdot K$ かつ $E_{B/A}$ が変数 V_i then

変数 V_i に $F \cdot K$ を代入し、 $W_{A/B}$ に K 個の V_i を変数拡張フラグとして入れ、 $W_{B/A}$ に K 個の n を入れ、Step 2 へ

else if E_A と E_B 両方とも変数 then

変数どうして代入して、Step 2 へ

case A/B が n かつ B/A が V_i の場合

$L_{A/B}$ の先頭から要素 $E_{A/B}$ を取る

if $E_{A/B} = F \cdot K$ then 変数 V_i に $F \cdot K$ を代入し、

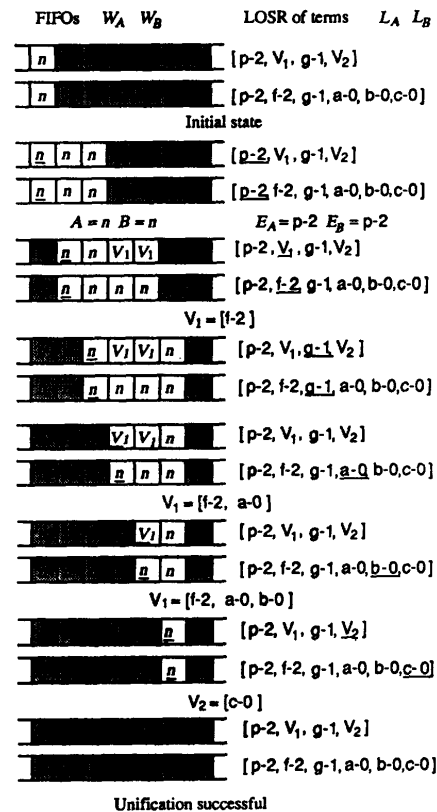


図 6 LOSR 上の単一化過程

Fig. 6 Unification process on LOSRs.

$W_{A/B}$ に K 個の n を入れ
 $W_{B/A}$ に K 個の V_i を入れ, Step 2 へ
else if $E_{A/B}$ が $F-K$ を代入された変数 V_j **then**
 変数 V_j に変数 V_i を代入し,
 $W_{A/B}$ に K 個の V_j を入れ,
 $W_{B/A}$ に K 個の V_i を入れ, Step 2 へ
else if $E_{A/B}$ が代入されていない変数 V_j **then**
 V_j に V_i を代入し, Step 2 へ
case A が V_i かつ B が V_j の場合
 変数どうして代入して, Step 2 へ ■

例として $[p-2, V_1, q-1, V_2]$ と $[p-2, f-2, q-1, a-0, b-0, c-0]$ の間の単一化の過程を図 6 に示す。

3.4 トライ構造のたどり方

項タプルを検索するために、前述した単一化のアルゴリズムをトライ構造をたどることに利用することを考える。トライ構造の各ノードを、アルゴリズム中の L_A の要素とし、検索条件の項の LOSR を L_B とする。トライ構造をたどる場合、分岐から単一化をやり直すことから、 L_A と L_B の途中から単一化が始められるようなバックトラック処理のための機構を用意する。

各変数はレベルを持つこととし、分岐を通るたびにその代入のレベルを一つずつ増やすようにする。変数代入の環境は、そのバックトラック・ポイントでの変数代入のレベルより高い代入内容を開放することにより元の状態に戻すことができる。図 6 の FIFO の中で、網かけのしていない部分が実際に作業領域として利用している部分であるが、これは先頭と最後を示すポイントによって表現することができる。最後に示すポイントが過ぎた後でも FIFO に格納された内容を残しておくことにより、そのポイントに戻すだけで FIFO の内容をバックトラックのポイントまで戻すことができる。結局、分岐のある要素 E_A が比較される場合、次のようなバックトラックのための情報をスタックに積む必要がある。

- E_A の分岐先の要素のポイント
- その時の検索条件の要素 E_B のポイント
- 変数代入のレベル
- W_A と W_B の先頭と最後を示すポイント

バックトラックが起こると、そのスタックの先頭の情報を取り出して、FIFO と変数代入の状況を分岐以前の状態まで戻してやり、スタックに積まれていた分

岐先のポイントと検索条件のポイントで示される要素の比較から単一化をしない。一方、トライ構造の葉の部分で、単一化が成功した場合には、その葉に対応する項タプルへのポイントとその時の変数代入の状況を結果として保存しておき、バックトラックを起こす。

このように、インデックスをたどっただけで、変数の代入内容まで求められるのは、この方法のもう一つの長所である。

4. 性能評価

C で書いた RBU のプロトタイプを使って、いくつかのタイプの項関係に対して、検索時間と挿入時間を測定し、ここで述べたインデックスの評価を行った。また、FOSR で表現された項に対する単一化処理の比較対象として、高速といわれている Quintus-Prolog のインタプリタについても同様の測定を行った。

次に挙げるような特徴を持つ四つのタイプの項関係を用意した。それぞれのタイプに対応するインデックスの形態を図 7 に示す。

タイプ A: すべての項が、最後の要素を除いてすべて等しい関数記号を持つ。インデックスは、たった一つのハッシュ・エントリに分岐が葉の部分にしかないような片寄った形の一つのトライ構造が接続される。

タイプ B: 先頭の要素が 16 の関数記号のうちのひとつを持ち、それ以外の要素は最後の要素を除いて等しい関数記号を持つ。インデックスは、タイプ A と同様な片寄った形の 16 個のトライ構造がハッシュ・テーブル

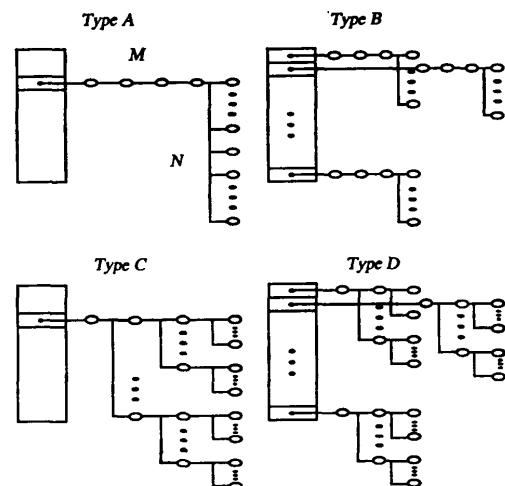


図 7 各タイプに対するインデックスの形態
Fig. 7 Index appearance for four type relations.

ルに接続される。この時、それぞれのトライ構造の大きさはほぼ同じとする。

タイプC: すべての項の先頭は同一の要素を持つが、それ以外の要素は再帰的に八つの関数記号のうちのどれかを持つ。インデックスは、一つのハッシュ・エントリに均整の取れた一つのトライ構造が接続される。トライ構造の各分岐は八つに分れる。

タイプD: 先頭の要素が 16 の関数記号のうち一つ

を持ち、それ以外の要素は再帰的に八つの関数記号のうちのどれかを持つ。インデックスは、タイプCと同様に均整の取れた 16 個のほぼ同サイズのトライ構造がハッシュ・テーブルに接続される。

これらのタイプに対して、50, 250, 500, 750, 1000 個の項タプルを持つ項関係を用意した。インデックスがある場合とない場合の RBU プロトタイプの単一化制約に要する時間、および同様の節集合に対する Prolog の節検索の時間の比較結果をタイプ A, B, C, D についてそれぞれ図 8~11 に示した。

図 8 は、3.2 節で述べた、比較の数が最も多くなる場合 (タイプ A) におけるトライ構造の効果を示したものである。この場合には、インデックスを張ってもエントリが一つであることからハッシングの効果はない。トライ構造の葉の部分で、タプル数分 (N) のバックトラックが起こるため、インデックスの有無にかかわらず、タプル数に比例して検索時間がかかる。インデックスがある場合とない場合の差は、比較回数が $N \times M$ から $N + M$ になったことによる。一つの項当たりの要素数が増えることによりこの差は広がる。

図 9 は、タイプ B の項関係に対する同様の測定結果である。これは、ハッシングによる絞り込みの効果を示しており、タプルの数が増すことに対する検索時間の増加がほとんどないことが分かる。

タイプ C の項関係を使った測定結果を示したのが図 10 である。この場合インデックスはトライ構造の効果だけとなるが、分岐によりバックトラックの数が大幅に削減されるため、検索時間がタプル数 (N) にまるで依存していないことが分かる。ハッシュ・テーブルの大きさにもよるが、この場合タイプ B より速くなっている。また、タイプ C にハッシングの効果を加えたのが図 11 であるが、タイプ C のみで十分に速いため、効果があまり現れていない。

このほか、更新処理におけるインデックス維持のオー

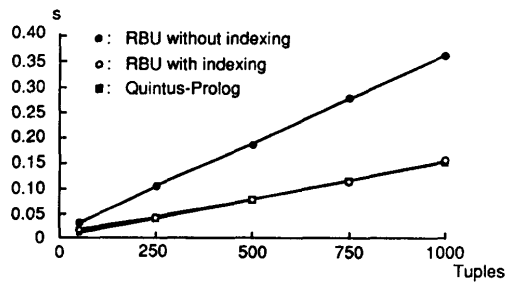


図 8 タイプ A に対する検索時間
Fig. 8 Search speeds for Type A.

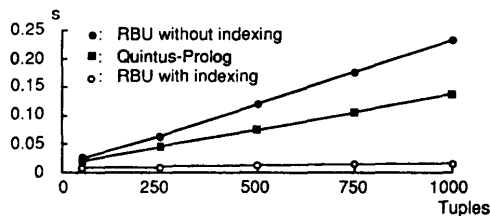


図 9 タイプ B に対する検索時間
Fig. 9 Search speeds for Type B.

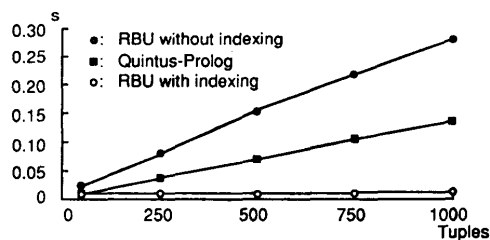


図 10 タイプ C に対する検索時間
Fig. 10 Search speeds for Type C.

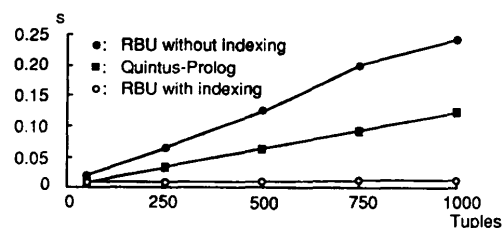


図 11 タイプ D に対する検索時間
Fig. 11 Search speeds for Type D.

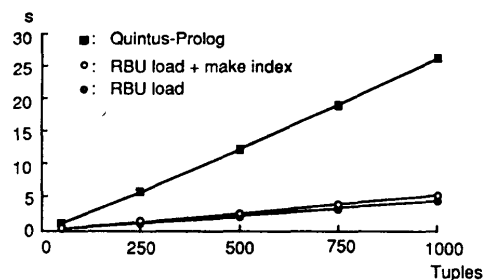


図 12 挿入時間の比較
Fig. 12 Insert speed comparison.

バヘッドを評価するための測定も行った。本インデックスはセル構造により実現され、構造が比較的単純であるため、挿入、削除操作を高速に行うことができる。図 12 に Prolog インタプリタとの挿入速度の比較と、インデックス作成にかかる時間を示した。インデックス作成が挿入の時間に対してわずかの割合でできることが分かる。なお、Prolog コンパイラは、コンパイル時間が長く、頻繁な更新を前提とする知識ベースシステムとは比較できない。

5. ま と め

RBU システムの高速化の手段として、トライ構造とハッシングを使ったインデックスを提案し、RBU 処理における要素間の比較処理とバックトラック処理が削減されることを示した。このトライ構造中で項を LOSR で表現することにより、速い段階で単一化可能な項を絞り込むことができる。我々は LOSR 用に FIFO を使った単一化アルゴリズムを提案すると共に、トライ構造をたどるためにスタックに積むべき内容を明らかにした。さらに、トライ構造とハッシングを組み合わせることにより、いろいろなタイプの項関係に対し効果的なインデックスを供給することができることを示した。

試作した RBU のプロトタイプシステムの単一化制約に要する時間と挿入に要する時間を各種項関係に対して測定した。明らかに、トライ構造とハッシングを使ったインデックスは、検索の高速化に非常に効果があり、更新におけるインデックス維持のオーバヘッドはわずかであることが分かった。一般に使われている FOSR の Prolog インタプリタと比較して十分速い検索、更新が可能である。

謝辞 日頃ご指導をいただく内田俊一 ICOT 研究部長、林弘富士通研究所情報処理研究部門長代理、ならびに試作に御協力いただいた富士通 SSL の山崎光彦氏、竹中伸一氏、北島由蔵氏に感謝します。

参 考 文 献

- 1) Yokota, H. and Itoh, H.: A Model and Architecture for a Relational Knowledge Base, *Proc. of the 13th International Symposium on Computer Architecture*, pp. 2-9 (1986).
- 2) Lloyd, J. W.: *Foundation of Logic Programming*, p. 124, Springer-Verlag (1984).
- 3) 小山, 田中: Definite Clause Knowledge Representation, *Proc. of the Logic Programming Conference '85*, pp. 95-106 (1985).

- 4) Ueda, K.: Guarded Horn Clauses, *Logic Programming '85*, Wada, E. (ed.), *Lecture Notes in Computer Science* 221, Springer-Verlag (1986).
- 5) Yokota, H., Kitakami, H. and Hattori, A.: Knowledge Retrieval and Updating for Parallel Problem Solving, *ICOT Technical Report*, TR-380 (1988).
- 6) Morita, Y., Yokota, H., Nishida, K. and Itoh, H.: Retrieval-By-Unification Operation on a Relational Knowledge Base, *Proc. of 12th International Conference on VLDB*, pp. 52-59 (1986).
- 7) Itoh, H., Takewaki, T. and Yokota, H.: Knowledge Base Machine Based on Parallel Kernel Language, *Proc. of 5th International Workshop on Database Machines*, pp. 15-28 (1987).
- 8) Ohmori, T. and Tanaka, H.: An Algebraic Deductive Database Managing a Mass of Rule Clauses, *Proc. of 5th International Workshop on Database Machines*, pp. 291-304 (1987).
- 9) Chang, C. L. and Lee, R. C. T.: *Symbolic Logic and Mechanical Theorem Proving*, p. 330, Academic Press (1973).
- 10) Ullman, J. D.: *Principles of Database Systems*, 2nd ed., p. 484, Computer Science Press, Potomac, Md. (1982).
- 11) Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, Sorting and Searching, p. 723, Addison-Wesley (1973).
- 12) Knuth, D. E.: *The Art of Computer Programming*, Vol. 1, Fundamental Algorithm, p. 634, Addison-Wesley (1973).

(平成 2 年 4 月 4 日受付)

(平成 3 年 1 月 11 日採録)

**横田 治夫 (正会員)**

1957 年生. 1980 年東京工業大学工学部電子物理工学科卒業. 1982 年同大学院情報工学専攻修士課程修了. 同年 4 月, 富士通(株)入社. 同年 6 月より(財)新世代コンピュータ技術開発機構に出向. 1986 年 4 月(株)富士通研究所に帰属. データベースマシン, 演繹データベース, 知識ベースシステム, 並列記号処理の研究・開発に従事. 電子情報通信学会, 人工知能学会各会員.

**北上 始 (正会員)**

1952 年生. 1976 年東北大学大学院修士課程修了. 同年富士通(株)入社. 1978 年～82 年(株)富士通研究所において関係データベース管理システムの研究・開発に従事. 1982 年 6 月～85 年 5 月(財)新世代コンピュータ技術開発機構において知識ベース管理システムの研究に従事. 1985 年 6 月(株)富士通研究所に帰属. 静止衛星の姿勢制御ソフトウェアの開発, 関係データベース管理システムの研究・開発, 知識ベース管理システムの研究. 情報処理学会 25 周年記念論文に採録. 日本認知学会, 日本ソフトウェア科学会, 人工知能学会各会員.

**服部 彰 (正会員)**

昭和 24 年生. 昭和 47 年大阪大学工学部電子工学科卒業. 昭和 49 年同大学院工学研究科修士課程修了. 同年(株)富士通研究所入社. 階層メモリ, 記号処理マシンの研究を経て, 並列コンピュータの研究開発に従事. 現在, 人工知能研究部長代理. 電子情報通信学会, 人工知能学会各会員.