# 連想メモリを利用したハードウェア向き単一化アルゴリズム†

大久保 雅且†† 安浦 寛人††*
高木 直史†† 矢島 脩三††

単一化 (unification) は，Prolog を始めとする論理型言語の基本操作として用いられており，その処理速度が Prolog プログラムの実行時間に及ぼす影響は大きい．しかし，ソフトウェアや専用マシンの Prolog 処理系で基本としている Robinson の単一化アルゴリズムは，複雑な問題に対して非常に効率が悪くなることが知られている．本論文では，ハードウェア向きの単一化アルゴリズムを提案する．本アルゴリズムは，単一化が UNION-FIND 問題に帰着できることに着目し，機能メモリの利用とパイプライン処理によって，高速に単一化を行う．特に，ここでは機能メモリとして，連想メモリ (Content Addressable Memory : CAM) を用いる方法を示す．CAM の，並列検索と並列書き込みの機能をうまく利用し，UNION-FIND 問題を高速に処理する．本アルゴリズムに基づくハードウェアは，ソフトウェアによる単一化と比較して 100 倍以上の速度が得られ，特に，複雑な単一化に対して有効であることが，ソフトウェアシミュレーションによって確認できた．CAM は集積度も高く，本方法は VLSI に適したアルゴリズムと考えられる．

## 1. はじめに

単一化 (unification) は，与えられた二つの項の最汎単一化置換 (most general unifier : mgu) を求める操作で，Robinson により導出の基本操作として導入された[1]．また，Prolog 等の論理型言語をはじめ，複雑な型構造を持つプログラム言語における型照合や，項書き換えシステム等，広汎な分野で利用されている[2]．特に，Prolog では最も基本となる操作であり，単一化の処理速度が Prolog プログラムの実行時間に及ぼす影響は大きい[3]．本論文では，ハードウェア向きの高速単一化アルゴリズムを提案し，その実現について考察する．

現在，Prolog 処理系で採用している単一化アルゴリズムは，ソフトウェア，専用マシンともに，文献 1) のアルゴリズムを基本としている．さらに，Prolog 専用マシンでは，マイクロプログラムの最適化やタグアーキテクチャ等による高速手法も併用されている[4]．しかし，文献 1) のアルゴリズムは複雑な単一化に対して非常に効率が悪くなることが知られている[5]．より効率的な単一化アルゴリズムも提案されてはいるが[2),5),6]，ハードウェアによる実現を考慮に入れたものは少ない．さらに，単一化は本質的に逐次的な性質を持っていると考えられ，並列化による著しい計算時間の短縮化は，一般に困難である[7),8]．

文献 6) に示されているように，単一化は，UNION, FIND という，集合に対する二つの操作によって処理できる．この二つの操作は UNION-FIND 問題として知られるが，これをソフトウェアによって効率良く処理するためには，複雑なデータ構造とパス圧縮等の操作が必要となる[9]．

本論文で提案するアルゴリズムでは，機能メモリの利用とパイプライン処理によって単一化を高速に処理する．特に，ここでは機能メモリとして，連想メモリ (Content Addressable Memory : CAM) を用いて UNION-FIND 問題を処理する方法を提案する．CAM の並列検索と並列書き込みの機能を利用し，単純かつ高速な処理が可能となる．さらに，項の読み出しと CAM に対する操作をパイプライン的に行うことにより，高速に単一化を行う．シミュレーションによる評価では，ソフトウェアによる単一化と比較して，100 倍以上の高速化が期待できる．

以下，2 章では準備として，本論文で扱う単一化について述べる．3 章では，単一化が UNION-FIND 問題に帰着できることを示し，パイプライン処理と機能メモリの利用によるハードウェア向きの単一化アルゴリズムを提案する．4 章では，CAM を利用した UNION-FIND 問題の処理手法を提案するとともに，本アルゴリズムに基づく単一化ハードウェアの構成法を述べる．5 章では，ソフトウェアシミュレーションによる性能評価を行う．

† A Hardware-Oriented Unification Algorithm Using a Content Addressable Memory by MASAAKI OHKUBO, HIROTO YASUURA, NAOFUMI TAKAGI and SHUZO YAJIMA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室
* 現在 京都大学工学部電子工学教室

## 2. 単 一 化

本論文では，第一階述語論理における単一化のみを考える．$F$ を関数記号の集合，$V$ を変数の集合とする．ただし，$F \cap V = \phi$ とする．各関数記号には引数の数が定まっており，0引数関数記号を特に定数という．なお，関数記号には，$f, g, a$ 等を，変数には，$X$, $Y$ 等を用いる．

$V \cup F$ 上の項 (term) を以下のように定義する．

(1) 変数，定数は項である．

(2) $t_1, t_2, \cdots, t_n$ を項，$f \in F$ を $n$ 引数関数記号とするとき，$f(t_1, t_2, \cdots, t_n)$ は項である． □

二つの項 $t_1, t_2$ が与えられたとき，$t_1\sigma = t_2\sigma$ となる最も一般的な置換（変数への項の代入）$\sigma$ を求める操作が単一化であり，このとき，この $\sigma$ を mgu (most general unifier) という[1]．

項は，項グラフにより表せる．項グラフは，同一の変数は一つの節点で表すという制約を持つ，ラベル付きの有向グラフである．なお，詳細な定義は文献7)に従う．項グラフ上の各節点が表す項を以下のように定義する．

(1) $X \in V$ をラベルとする節点は，項 $X$ を表す．

(2) $n$ 引数関数記号 $f \in F$ をラベルとする節点は，$f$ の $i$ 番目の出力枝の先の節点がそれぞれ項 $t_i$ を表すとき，項 $f(t_1, t_2, \cdots, t_n)$ を表す． □

関数記号をラベルとする節点を関数節点，変数をラベルとする節点を変数節点と呼ぶ．項グラフを図で示す場合には，関数節点を丸で囲み，各節点を一意に識別するために節点番号を付ける．また，各節点から出る枝を，左から順に1番目，2番目，…の枝と呼ぶ．

[例題1]

$t_1 = f(X_1, X_2, a, X_1)$

$t_2 = f(a, g(Y_1), Y_2, Y_2)$

に対する項グラフは，**図1**のようになる．図1の〈1〉，〈5〉の節点がそれぞれ項 $t_1, t_2$ を表す．また，mgu は $\{a/X_1, g(Y_1)/X_2, a/Y_2\}$ となる．ただし，$t/X$ は，変数 $X$ を項 $t$ に置換することを表す． □

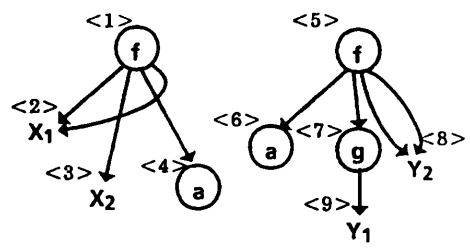



**図 1** 例題1に対する項グラフ
Fig. 1 Term graph for Example 1.

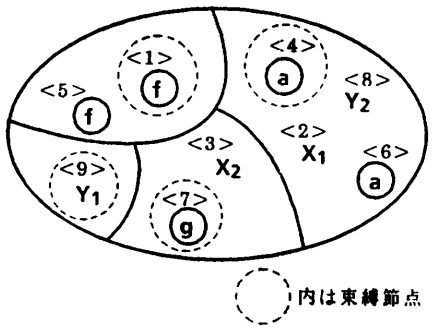



**図 2** 例題1に対する最小同値分割
Fig. 2 Minimum equivalent partition for Example 1.

項グラフの節点の集合を $N$ とするとき，次の条件を満たす $N$ の分割を同値分割という．(ただし，分割の要素をクラスと呼び，$i$ 番目のクラスを $c_i$ としたとき，$i \neq j$ で，$c_i \cap c_j = \phi$，$\bigcup_i c_i = N$ である．)

条件1 各クラスに含まれる関数節点のラベルはすべて等しい．

条件2 一つのクラスに二つ以上の関数節点が含まれる場合には，それらの対応する子供の節点どうしは，それぞれ同じクラスに属する． □

$t_1, t_2$ を表す節点が同じクラスに属するような $N$ の同値分割が存在するとき，$t_1$ と $t_2$ は単一化可能で，最も細かい同値分割（最小同値分割）が mgu に対応する．例題1に対する最小同値分割は**図2**のようになる．

最小同値分割から，mgu，および，単一化された項は容易に求められる[10]．すなわち，単一化は，与えられた二つの項の項グラフに対して単一化可能かどうかを判定し，単一化可能な場合には最小同値分割を求めればよい．

## 3. 単一化のアルゴリズム

### 3.1 基本アルゴリズム

与えられた $t_1, t_2$ に対する最小同値分割は，次の方針で求めることができる．

(1) 項グラフの各節点は，各々「その節点のみを要素とするクラス」を形成している状態を初期状態とする．

(2) $t_1, t_2$ を表す節点を同じクラスに入れる．

(3) 同値分割の条件に合うようにクラスどうしの

併合を繰り返す.

上記の操作を行うためには，各クラスに名前を付けると都合がよい．ここで，変数節点のみから成るクラスはそのうちの一つの変数節点を，関数節点を含むクラスは一つの関数節点を，それぞれ，そのクラスの束縛節点と呼び，これを各クラスの名前として利用する(図2参照).

節点 $t_1$ が表す項と節点 $t_2$ が表す項の単一化アルゴリズムを以下に示す．ただし，BIND($t$) は節点 $t$ が属するクラスの束縛節点を，MERGE($t_1, t_2$) は節点 $t_1$ が属するクラスと節点 $t_2$ が属するクラスの併合を，それぞれ表す.

単一化の基本アルゴリズム

step 1　BIND($t_1$)≠BIND($t_2$) ならば step 2 以下の操作を行う．(BIND($t_1$)=BIND($t_2$) ならば何もしない)

step 2　BIND($t_1$) または BIND($t_2$) が変数節点ならば，MERGE($t_1, t_2$) とする．BIND($t_1$) と BIND($t_2$) がともに関数節点の場合には，step 3 から step 5 までを行う.

step 3　BIND($t_1$) と BIND($t_2$) のラベルとが異なれば，単一化不可能として終了する.

step 4　MERGE($t_1, t_2$)

step 5　BIND($t_1$) と BIND($t_2$) の，すべての対応する子供の節点どうしについて，step 1 以下を行う.

次に，上記の基本アルゴリズムを実現するためのハードウェアアルゴリズムについて述べる.

### 3.2 単一化のハードウェアアルゴリズム

3.1 節で示したアルゴリズムは，BIND，MERGE 等の操作による最小同値分割の形成 (step 1〜step 4) と，項グラフの探索 (step 5) の，二つの処理に分けられる．さらに，step 5 で求めたすべての節点対について step 1 以下の操作を行うため，両者はパイプライン処理が可能である．そこで，3.1 節のアルゴリズムを実現するハードウェアとして，**図3**を考える.

図3の項グラフ処理部は，アルゴリズムの step 5 に対応する．FIFO 2 より関数節点対を取り出し，それらの対応する子供の節点対をメモリから読み出して，順次 FIFO 1 に送り出す.

また，単一化処理部では，FIFO 1 から節点対を取り出し，機能メモリを利用して，BIND と MERGE の二つの操作を行う．束縛節点はクラス名として利用しているので，これらの操作は，節点の属するクラス名の検索 (FIND) と，二つのクラスの併合 (UNION) と考えられ，UNION-FIND 問題[9]に帰着できる．したがって，機能メモリでは，節点の分割の状態を記憶しておき，UNION-FIND 問題を高速に処理できればよい．具体的には，以下の機能が必要となる.

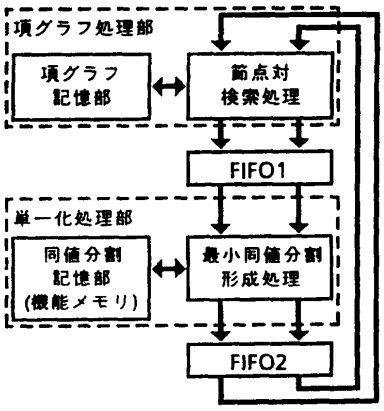



**図3** 単一化用ハードウェアのブロック図
Fig. 3 Block diagram of unification hardware.

BIND……入力：節点 $t$
　　出力：$t$ が属するクラスの束縛節点

MERGE…入力：節点 $t_1, t_2$
　　処理：$t_1, t_2$ が属するクラスの併合

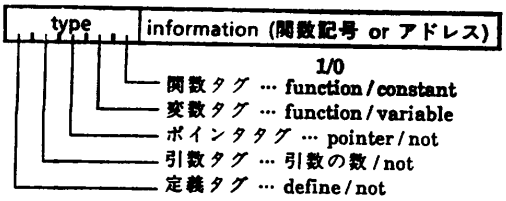
これらが，問題の大きさ（節点の数）にかかわらず，一定の処理時間（クロック数）で行えれば，図3のパイプラインは停滞することなく処理が行える．4章では，この機能メモリが CAM を用いて実現できることを示す.

## 4. ハードウェアによる実現法

### 4.1 項グラフの内部表現

項グラフ記憶部では，タグとポインタを利用して項グラフを記憶する（**図4**参照）．節点のラベルの種類を表すために，変数タグ，定数タグを利用する．引数タグが '1' の語は，すぐ上の語の関数節点の出力枝の数を表す．4.2 節で示すように，単一化処理部の CAM では，初期状態としてすべての語の内容を 0 としておき，以後の操作によって束縛節点を書き込む．CAM の語の内容が，初期状態のままであるか，束縛節点を記憶しているかを判断するために定義タグを用いる．例題1の項グラフの内部表現は図4(b)のようになる．図4(b)の太枠で囲んだ部分が節点のデータとなる.

また，本方法では，項グラフ上の節点を一意に識別する必要がある．図1では「論理的な」節点番号を付




(a) 各語の内容
(a) Contents of Word

| ad. | type | inform. |
|---|---|---|
| 1 | 10011 | f |
| 2 | 11000 | 4 |
| 3 | 10000 | $X_1$ |
| 4 | 10000 | $X_2$ |
| 5 | 10010 | a |
| 6 | 10100 | 3 |

| ad. | type | inform. |
|---|---|---|
| 7 | 10011 | f |
| 8 | 11000 | 4 |
| 9 | 10010 | a |
| 10 | 10100 | 13 |
| 11 | 10000 | $Y_2$ |
| 12 | 10100 | 11 |
| 13 | 10011 | g |
| 14 | 11000 | 1 |
| 15 | 10000 | $Y_1$ |

(b) 図1の項グラフの内部表現
(b) Internal Expression for Term Graph of Fig.1

図 4 項グラフの内部表現
Fig. 4 Internal expression of term graph.

けたが，実際には，節点を記憶している語のアドレスを節点番号として利用すればよい．これにより，関数節点の番号，すなわち，そのアドレスの入力によってその節点の子節点を容易に求められる．また，FIFO 1には，節点のデータと節点番号をともに格納する．

### 4.2 単一化処理部

3.2 節で述べたように，アルゴリズムの step 1 から step 4 は UNION-FIND 問題に帰着できる．ここでは，まず，CAM を利用して一般の UNION-FIND 問題を処理する手法を提案し，次に本アルゴリズムへの適用を考える．

#### 4.2.1 UNION-FIND 問題の高速処理手法

UNION-FIND 問題を定式化すると，集合に対する次の二つの操作となる．

FIND ($i$)…要素 $i$ が属する集合の集合名の検索

UNION ($i, j$)…要素 $i$, 要素 $j$ が属する集合の併合

ただし，異なる二つの集合は共通の要素を含まないものとする．

CAM は，内容による一致検索，および，一致した語へのデータの書き込み等を行えるメモリである．本論文では，

（1） アドレスによる読み書き（RAM 機能）
（2） 検索データによる並列一致検索
（3） 一致した語への並列書き込み
（4） 検索データのマスク

といった四つの機能を持つ CAM の利用を考える．これらの機能を有する CAM には，例えば文献11)が挙げられる．

CAM には，アドレスに要素番号を対応させ，アドレス $i$ の語の内容(以下 CAM[$i$] と記す)として要素 $i$ の属する集合名を記憶する．また，初期状態として，各要素 $i$ は集合 $i$ に属するものと考え，CAM のすべての語の内容を 0 としておき，以後の UNION 操作によって新しい集合名に書き換えてゆく．CAM を利用した FIND, UNION は，次の操作となる(図 5 参照)．

FIND ($i$)…CAM[$i$] の読み出し
もし 0 ならば，$i$ とする．

UNION ($i, j$)…**step** 1 $R_1 \leftarrow$ CAM[$i$]
step 2 $R_2 \leftarrow$ CAM[$j$]
step 3 $R_1, R_2$ の内容による分岐
case 1 $R_1 \neq 0$, $R_2 \neq 0$
step 4 検索レジスタ$\leftarrow R_2$
step 5 一致検索
step 6 一致した語$\leftarrow R_1$
case 2 $R_1 \neq 0$, $R_2 = 0$
step 4 CAM[$j$]$\leftarrow R_1$
case 3 $R_1 = 0$, $R_2 \neq 0$
step 4 検索レジスタ$\leftarrow R_2$
step 5 一致検索
step 6 一致した語$\leftarrow i$
step 7 CAM[$i$]$\leftarrow i$
case 4 $R_1 = 0$, $R_2 = 0$
step 4 CAM[$i$]$\leftarrow i$
step 5 CAM[$j$]$\leftarrow i$

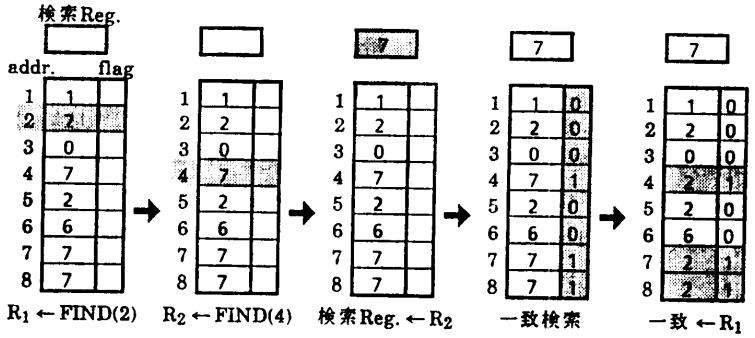



図 5 UNION (2, 4) のときの CAM の操作
Fig. 5 Operations of CAM for UNION (2, 4).

CAM では，並列検索，並列書き込みが可能なため，上記の各ステップは1クロックで行える．したがって，FIND は1～2クロックで，UNION は4～7クロックで完了する．なお，CAM の初期化（すべての語の内容を0とする）は，

step 1　検索データのすべてのビットをマスク
step 2　一致検索（すべての語が選択される）
step 3　一致した語← 0

といった3クロックで行える．

#### 4.2.2 単一化処理部

FIFO 1 より節点対を取り出し，CAM を利用して BIND, MERGE 等の操作を行う．CAM のアドレスに節点番号を対応させ，また，CAM[$i$] に節点番号 $i$ の節点の属するクラスの束縛節点とその節点番号を記憶する．4.2.1 項で示した UNION$(i, j)$ では，UNION 後の集合名を要素 $i$ の属する集合名に統一したが，MERGE$(t_1, t_2)$ 後のクラス名は，束縛節点にする必要がある．したがって，MERGE のときのクラス名は，

（1） BIND$(t_1)$ が変数節点かつ BIND$(t_2)$ が関数節点のとき，BIND$(t_2)$
（2） 上記以外のとき，BIND$(t_1)$

となるように，CAM を操作する．すなわち，4.2.1 項で示した操作中の $i, j, R_1, R_2$ の内容によって，検索データ等を決定する．また，束縛節点がともに関数節点の場合には，その節点番号対を FIFO 2 に格納する．

### 4.3 例　　題

例題1の処理過程における CAM の内容を図6に示す．

（a） 初期状態（CAM のすべての語の内容を0とする）
（b） $t_1$ を表す節点と $t_2$ を表す節点を同じクラスに入れる．両者とも関数節点なので，それらが項グラフ処理部にフィードバックされる．
（c）～（f） 項グラフ処理部では，対応する子供の節点の読み出しが行われ，次々に単一化処理部に送られる．単一化処理部では，送られてきた節点対に対し，BIND, MERGE 等の操作により最小同値分割を形成していく．これらの二つの処理はパイプライン的に行われる．

図6の，（b）～（d）は case 4 に，（e）は case 2 に，（f）は case 1 に対応して，それぞれ，CAM に対する操作が行われる．図2，図6から明らかなように，処理終了時の CAM の内容（図6の（f））が最小同値分割となる．

## 5. 評　　価

### 5.1 シミュレーションによる評価

図3のハードウェアの，実際の単一化の問題に対する処理時間を，ソフトウェアシミュレーションによって評価した．シミュレーションでは，メモリ（RAM, CAM, FIFO）へのアクセス，比較器による比較や条件判断等を1クロック内で完了すると仮定した．以下に使用した例題を示す．

[例題2]

$t_1 = f(X_1, X_2, \cdots, X_n)$
$t_2 = f(a_1, a_2, \cdots, a_n)$

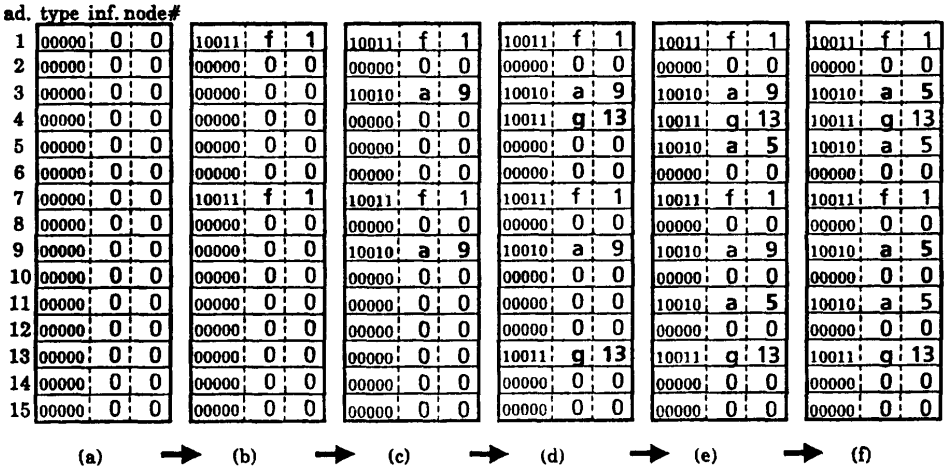



図 6　例題1に対する CAM の操作
Fig. 6　Operations of CAM for Example 1.

表1 シミュレーション結果
Table 1 Result of simulation.

| | | $n=5$ | $n=10$ | $n=20$ | $n=25$ |
|---|---|---|---|---|---|
| 例題2 | 本方法(クロック数) | 33 | 58 | 108 | 133 |
| | 本方法(換算値)† | 0.0066 | 0.0116 | 0.0216 | 0.0266 |
| | *CP* | 1.22 | 1.98 | 3.52 | 4.32 |
| | *JJ* | — | 1.60 | — | 3.74 |
| 例題3 | 本方法(クロック数) | 96 | 196 | 396 | 496 |
| | 本方法(換算値)† | 0.0192 | 0.0392 | 0.0792 | 0.0992 |
| | *CP* | 6.60 | 144 | $1.43\times10^{3}$ | $4.40\times10^{4}$ |
| | *JJ* | 3.57 | 7.35 | — | 18.9 |

注) 単位: ms
†: 1クロックを 200 ns とする

[例題3]

$t_1=f(g(X_1, X_1), g(X_2, X_2), \cdots, g(X_{n-1}, X_{n-1}), Y_2, Y_3, \cdots, Y_n, Y_n)$

$t_2=f(X_2, X_3, \cdots, X_n, g(Y_1, Y_1), g(Y_2, Y_2), \cdots, g(Y_{n-1}, Y_{n-1}), X_n)$

例題2は，単純な単一化で，アルゴリズムを実現した際のオーバヘッドを評価できる[5]．また，例題3は，複雑な単一化で，単一化アルゴリズムの評価にしばしば用いられる[5]．

シミュレーション結果を表1に示す．表1では，処理に必要なクロック数を示すとともに，1クロックを200 ns とした場合の処理時間を示した．比較データとして示した *CP* は，vax 11/750 上の c-prolog による実測値で，節起動等のオーバヘッドを除いた単一化のみの処理時間である．*JJ* は，「COMMON」と「FRONTIER」を求めて単一化を行うアルゴリズムの vax 11/780 上での処理時間で，文献 5) より引用した．

### 5.2 考　　察

表1では1クロックを 200 ns としたが，文献 11) の CAM が 140 ns でアクセスでき，また，FIFO や RAM も同程度の時間でアクセス可能と考えられるため，この仮定は妥当と考えられる．

例題2に対して，*JJ* と *CP* は同程度の処理時間であるが，使用した計算機の処理速度の差（カタログ値で約1.7倍）を考慮すると，*JJ* はオーバヘッドが少なくない．これに対し，本方法はオーバヘッドも少なく，処理速度も両者の 100〜200 倍となる．しかし，高速の Prolog 処理系も発表されており[12]，これらと比較した場合，本方法が特に有効かどうかは断定できない．

```
p1(X2) :- p2(g(X2, X2)).
p2(X3) :- p3(g(X3, X3)).
    ....
pn-2(Xn-1) :- pn-1(g(Xn-1, Xn-1)).
pn-1(Xn) :- pn(g(Y1, Y1), Xn).
pn(Y2, Xn) :- pn+1(g(Y2, Y2), Xn).
pn+1(Y3, Xn) :- pn+2(g(Y3, Y3), Xn).
    ....
p2n-3(Yn-1, Xn) :- p2n-2(g(Yn-1, Yn-1), Xn).
p2n-2(Yn, Xn) :- p2n-1(Yn, Xn).
p2n-1(Xn, Xn).
```

図7 例題3と等価な Prolog のプログラム
Fig. 7 Equivalent Prolog program for Example 3.

例題3のような複雑な単一化の場合には，本方法は特に有効である．すなわち，Prolog 処理系で基本としている文献1)の単一化アルゴリズムでは，この例題に対し処理時間は $2^n$ に比例して増加するが，本方法では $n$ に比例した処理時間となる．また，*JJ* も効率的なアルゴリズムであるが，本方法は *JJ* の 200 倍近い速度が得られる．

これらの考察より，本方法はオーバヘッドも少なく，また，複雑な単一化に対して特に有効と考えられる．

実際の Prolog のプログラムでは，例題3のような複雑な単一化が明示的に現れることは少ないと考えられる．しかし，図7のプログラムに対して，

?- $f_1(g(X_1, X_1))$

というゴール節を入力した場合，例題3と等価な単一化が行われる．図7のそれぞれの節 (clause) は，引数の数や構造体の出現数等，Prolog プログラムの平均[13]と比較しても複雑ではない．すなわち，一見単純な Prolog のプログラムでも，複雑な単一化が行われることは十分予想され，単純な単一化から複雑な単一化まで高速に処理できる本方法は有効と考えられる．

## 6. おわりに

ハードウェア向きの単一化アルゴリズムを提案した．本アルゴリズムでは，単一化が UNION-FIND 問題に帰着できることに着目し，機能メモリの利用とパイプライン処理によって高速に単一化を行う．4章で示した実現法では，機能メモリとして CAM を用いる手法を提案した．5章で評価したように，本方法はオーバヘッドも少なく，特に，複雑な単一化に対して有効である．現在，高集積化された CAM も発表されていることより，本方法は，実現性も高く，また，VLSI 化に適していると考えられる．さらに，本論文

で提案した UNION-FIND 問題の CAM を利用した処理手法は，単純かつ高速であり，単一化以外にも，UNION-FIND 問題を基本とする様々な問題[9]に対して，応用できる．

項グラフは，多くの Prolog 処理系では既に利用されてはいるが，これを明示的には記憶していない．処理系によって，構造共有法 (structure sharing)，あるいは，構造複写法 (structure copy) を用いて，項グラフを「ばらばらに」記憶している．また，「環境」の下での変数の評価等，本方法を Prolog に組み込むためには何らかの整合を取る必要があろう．4章で示した項グラフの内部表現は一例であり，各処理系で採用している記憶管理法に従って，柔軟な対応ができると考えられる．このとき，節点番号として，節点のアドレスを利用できるかどうかによって，組み込みやすさが決まると考えられる．

今後は，本方法を実際の Prolog 処理系に組み込む際の問題点等について検討したい．

**謝辞** 御討論いただいた本学情報工学教室矢島研究室の諸氏，および，c-prolog の性能評価に御協力いただいた同教室萩原研究室の皆様に感謝いたします．また，単一化に関する資料を呈示いただいた ICOT の諸氏，および CAM の利用に関して御討論いただいた NTT 電気通信研究所小倉武氏，長沼次郎氏ほか集積応用研究室の皆様に深謝いたします．本研究は一部文部省科学研究費補助金による．

## 参 考 文 献

1) Robinson, J. A.: A Machine-Oriented Logic Based on the Resolution Principle, *J. ACM*, Vol. 12, No. 1, pp. 23-41 (1965).
2) Martelli, A. and Montanari, U.: An Efficient Unification Algorithm, *ACM TOPLAS*, Vol. 4, No. 2, pp. 258-282 (1982).
3) Nam Sung Woo: A Hardware Unification Unit: Design and Analysis, *12th Annual Int. Symp. on Computer Architecture*, pp. 198-205 (1985).
4) Yokota, M., Yamamoto, A., Taki, K., Nishikawa, H. and Uchida, S.: The Design and Implementation of a Personal Sequential Machine: PSI, *New Generation Computing*, Vol. 1, pp. 125-144 (1983).
5) Jaffar, J.: Efficient Unification over Infinite Terms, *New Generation Computing*, Vol. 2, pp. 207-219 (1984).
6) Paterson, M. S. and Wegman, M. N.: Linear Unification, *J. CSS*, Vol. 16, pp. 158-167 (1978).
7) Yasuura, H.: On Parallel Computational Complexity of Unification, *FGCS '84*, pp. 235-243 (1984).
8) Dwork, C., Kanellakis, P. C. and Mitchell, J. C.: On the Sequential Nature of Unification, *J. Logic Programming*, Vol. 1, pp. 35-50 (1984).
9) Aho, J. E., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).
10) 安浦寛人，大久保雅且，矢島脩三：論理型言語の単一化操作のためのハードウェアアルゴリズム，信学技報，EC 84-67 (1985).
11) 小倉 武，山田慎一郎，丹野雅明，石川浩司：4 Kb CMOS 連想メモリ LSI，信学技報，SSD 83-78 (1983).
12) 浅川康夫，田村直之，小松秀昭，黒川利明：複数のアーキテクチャをターゲットにした高速 Prolog コンパイラ，情報処理学会記号処理研究会資料，86-37 (1986).
13) 尾内理紀夫，清水 肇，益田嘉直，麻生盛敏：逐次型 Prolog プログラムの解析，*Proc. Logic Programming Conf. '84*, 9-5 (1984).

(昭和 61 年 11 月 11 日受付)
(昭和 62 年 6 月 11 日採録)

**大久保雅且**（正会員）

昭和 36 年生．昭和 60 年京都大学工学部情報工学科卒業．昭和 62 年同大学院修士課程修了．現在，日本電信電話(株) NTT ヒューマンインタフェース研究所勤務．在学中はハードウェアアルゴリズムの研究に従事．電子情報通信学会会員．

**安浦 寛人**（正会員）

昭和 28 年生．昭和 51 年京都大学工学部情報工学科卒業．昭和 53 年同大学院修士課程修了．昭和 55 年 4 月より京都大学工学部情報工学教室助手，昭和 61 年 11 月，京都大学工学部電子工学教室助教授となり現在に至る．工学博士．非同期回路，論理回路の複雑さ，VLSI 向きハードウェアアルゴリズム，論理設計の CAD 等の研究に従事．IEEE，電子情報通信学会，EATCS，LA シンポジウム，日本ソフトウェア科学会各会員．

**高木 直史**（正会員）

昭和34年生．昭和56年京都大学工学部情報工学科卒業．昭和58年同大学院修士課程修了．昭和59年4月より京都大学工学部助手．VLSI向きハードウェアアルゴリズム，算術演算回路，論理設計用 CAD/DA 等の研究に従事．IEEE，電子情報通信学会各会員．

**矢島 脩三**（正会員）

昭和8年生．昭和31年京都大学工学部電気工学科卒業．同大学院博士課程修了．工学博士．昭和36年より京大工学部に勤務，昭和46年情報工学科教授．昭和35年京大第一号計算機 KDC-1 を設計稼動，以来，計算機，論理設計，オートマトン等の研究教育に従事．著書は「電子計算機の機能と構造」（岩波，57年）等．本学会元常務理事，元会誌編集委員（地方），元 JIP 編集委員．電子情報通信学会元評議員およびオートマトンと言語研専元委員長，North-Holland 出版 IPL 編集委員，IEEE Senior Member.