Prologプログラムの C への変換

片峯恵一[†] 廣田豊彦[†]
周能法[††] 長澤勲[††]

ドメインの知識を記述するための言語処理系のプロトタイプ用言語には，1) 言語処理系の開発に適していて，理解性，保守性にすぐれている，2) 実際に運用可能な性能を実現できる，3) 移植性にすぐれている，などの要件がある．そのような要件を満たす言語として著者らは β-Prolog の開発を進めている．本研究では，決定的な述語を C へ変換することによって，β-Prolog のいっそうの高速化を目指した．具体的な変換手順は，1) β-Prolog コンパイラの中間データである照合木を入力とする，2) 照合木を抽象機械 NTOAM の命令系列へ展開する，3) NTOAM の各命令を C の文へ変換する，となる．このようにして C へ変換された決定的な述語は，NTOAM 上で解釈される非決定的な述語と組み合わせて実行することができる．実行性能を評価した結果，非決定性を含む 8-queen のような問題であっても，SICStus のネイティブコード・コンパイラよりも高い性能を示した．人手で C プログラムへ変換することによってさらに性能を向上させることが可能であるが，そのためには膨大な労力を要し，しかも理解性や保守性が低下する．したがって，本論文で提案した手法は，実用的なプロトタイプの開発にきわめて有用であると考えられる．

On the Translation of Prolog Programs to C

KEIICHI KATAMINE,[†] TOYOHIKO HIROTA,[†] NENG-FA ZHOU[††]
and ISAO NAGASAWA[††]

The requirements for a language to be a good implementation language for knowledge representation languages include: 1) be suitable for developing language processors, and have good understandability and maintainability, 2) have fair performance tolerable for practical uses, and 3) have good portability. We have been developing β-Prolog to satisfy the above requirements. In this research, we have aimed to improve the performance of β-Prolog by translating determinate predicates to C. Our translation procedure works as follows: 1) input the matching trees which are intermediate data of the β-Prolog compiler, 2) flatten the matching trees into a sequence of NTOAM instructions, 3) convert each NTOAM instruction to C statements. The C program translated from determinate predicates can be linked with nondeterminate predicates which are interpreted by NTOAM. Our evaluation result shows that our method is superior in performance to SICStus native code compiler even when a program has nondeterminism like 8-queen. Manual translation to C program might improve the performance further, but it would take enormous efforts and degrade the understandability and maintainability. We think that our method is much effective for developing practical prototype systems.

1. はじめに

知的設計支援システムを開発するためには，設計対象の知識を分析，整理し，記述することが不可欠である．これに対して従来のソフトウェア工学では，OMT (Object Modeling Technique)[1]などのオブジェクト指向方法論や，各種の CASE (Computer Aided Software Engineering) ツールなど，汎用性の高いものが提案され，使用されてきた[2]．

設計対象の知識はドメイン特有のものであるため，そのドメインの専門家が直接記述することが望ましいが，ドメインの専門家が上述のような汎用性の高い方法論やツール，言語を使いこなすことは容易ではない．そこで知識を記述するための概念モデル記述言語をドメインに特化し，ドメインの専門家に使いやすいもの

† 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
†† 九州工業大学情報工学部機械システム工学科
Department of Mechanical Systems Engineering, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology

にする必要がある[3),4)].

著者らはそのような記述言語として，機械設計などの設計計算を支援する DSP[5)]や，建築物の構造設計を支援する言語 BDL[6)]などの開発を進めている．このような言語の成功の鍵は，いかに専門家の意見を反映するかという点にあり，そのために当初から言語仕様を確定することは困難である．プロトタイプを作成し，それをドメインの専門家に使ってもらうというプロトタイピングのサイクルを反復する必要がある．

以上のようなことから，ドメインの知識を記述するための言語処理系のプロトタイプ用言語には，1) 言語処理系の開発に適していて，理解性，保守性などにすぐれている，2) 実際に運用可能な性能を実現できる，3) 移植性にすぐれている，などの要件があると考えられる．そのような要件を満たす開発言語として著者らは β-Prolog[7)~9)]の開発を進めている．

β-Prolog では照合木指向の抽象機械 NTOAM を用いることによってユニフィケーションの高速化をはかっている．本研究では，決定的な述語を C へ変換することによってよりいっそうの高速化を目指した．その結果，移植性や理解性など，ソフトウェア工学的に重要な性質を犠牲にすることなく，きわめてすぐれた実行性能を得ることができた．

本論文では，Prolog プログラムから C への変換手法について述べ，処理性能やプログラムの理解性などについて評価する．以下，2 章で β-Prolog について簡単に紹介する．3 章で本研究の Prolog から C への変換手法について述べる．4 章で本研究の変換手法の評価を述べ，5 章で関連研究について言及する．

2. β-Prolog

本研究の対象とした β-Prolog について，特に C への変換にかかわる照合木ならびに NTOAM について説明する．

2.1 β-Prolog の概要

β-Prolog は高速性と移植性を目的とした Prolog 処理系であり，以下のような特徴を持っている．

- NTOAM (New matching Tree Oriented Abstract Machine)[9)]エミュレータをベースとした処理系であり，移植性が高い．NTOAM は WAM (Warren's Abstract Machine)[10)]を継承しているが，照合木を用いたユニフィケーションの高速化，パラメータ受渡しの高速化，などの特徴を持っている．
- DEC-10 Prolog におけるエジンバラスタイルの節[11)]のほかに，照合節形式[7)]☆を持っている．照合節は，入力と出力のユニフィケーションを分離し，さらに決定性を明示する．利用者はエジンバラスタイルでプログラムを書くことができるが，照合節で書くことによって，コンパイラはより効率的なプログラムを生成することができる．
- 状態表という特殊なデータ構造[8)]を持っており，制約充足問題や組合せ探索問題を高速に解くことができる．

2.2 照 合 木

照合木は，Prolog プログラムから NTOAM の命令を生成するための中間データ構造であり，根 (root)，テストノード (test node)，葉 (leaf) の 3 要素から構成される．

根は述語名を表しており，ヘッドの述語名が同じであるようなすべての節は 1 つの照合木に統合される．

テストノードはユニフィケーションを含まない処理を表す．ヘッド述語の引数の型のチェックや照合節形式におけるガード☆☆の処理が行われる．テストノードの条件が成立しないときには，次のテストノードを試すことになる．次のテストノードが存在しないときには，この照合木が表す述語の実行が fail となる．いずれの場合でも，ユニフィケーションを行っていないので，バックトラックの処理が簡単になる．β-Prolog ではこれをシャローバックトラック (shallow backtrack) とよんでいる．

葉はボディ述語の処理に対応する．ここで処理が fail となると，変数の束縛を復旧するディープバックトラック (deep backtrack) が発生する．

従来の WAM の処理では，述語が呼び出されると，引数について順にユニフィケーションを行う．そして，途中の引数でユニフィケーションに失敗するとディープバックトラックが発生する．それに対して，照合木上では，まずすべての引数の型のチェックが行われ，その後にユニフィケーションが行われるので，ディープバックトラックの回数を減らせる可能性がある．

例として**図 1** に示す述語 **membchk/2** の照合木を**図 2** に示す．WAM では第 1 引数がユニファイされた後に第 2 引数がリストであるかどうかチェックされるので，たとえば，それが空リストのときには必ずディープバックトラックが行われる．一方照合木では，まずテストノードで第 2 引数がリストかどうかがテストされるので，リストでないときでもシャローバック

☆ 以前の文献では standard form (標準形) とよんでいた．
☆☆ ユニフィケーションを含まない処理だけが記述できる．

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

図1 membchk/2
Fig. 1 membchk/2.

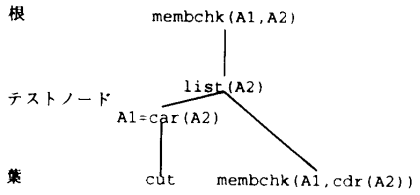

図2 membchk/2の照合木
Fig. 2 Maching tree of membchk/2.

```
lab0: jmpn_list   Y2   lab2
      fetch       X1
      fetch       Y3
      jmpn_id     Y1   X1   lab1
      return
lab1: move        Y2   Y3
      jmp         lab0
lab2: fail
```

図3 membchk/2のNTOAMコード
Fig. 3 NTOAM code of membchk/2.

トラックですむことがある☆.

2.3 NTOAM

NTOAMはWAMをベースとした仮想機械であるが，前節で述べた照合木の形式のプログラムを効率良く実行するためにいくつかの命令が追加されている．代表的な命令としては，テストノードの実行を行う各種の条件分岐命令がある．またシャローバックトラックとディープバックトラックの2種類のバックトラックを扱うためのレジスタや命令を持っている．

NTOAMのもう1つの特徴は，パラメータをレジスタではなく，スタック上で受渡しすることである．これによって，他の述語を呼び出すときに，自身のパラメータの保存・復旧をする必要がない．また，尾部再帰を反復に置き換えたとき，スタック上のパラメータをそのまま再利用することが可能になる．

図1に示した述語 **membchk/2** のNTOAMコードを**図3**に示す．まず **jmpn_list** で，第2引数（Y2）がリストであるかどうかをチェックする．リストではないときには **lab2** へ分岐し，**fail** を実行する．第2引数がリストのときには，その頭部と尾部を取ってくる．頭部が第1引数（Y1）と一致するときには **return** を実行し，そうでないときには，尾部を第2引数として自分自身を再度実行する．

3. Cへの変換

3.1 変換の基本方針

PrologからCへの変換の手法としては，WAMに基づく手法[12]を基本とした．その主な理由は，

(1) β-PrologはWAMをベースにした仮想機械NTOAM上で実行される．
(2) NTOAMのインタプリタはC言語によって書かれている．

である．この方式を用いてβ-Prologプログラムを実行するときのプログラム変換の流れを**図4**に示す．図中の「Prolog-to-C変換系」の部分が本研究で提案する変換を実現するプログラムである．

NTOAMプログラムからCプログラムへの変換は，NTOAMの命令語に対して，Cで書かれたインタプリタと同様のCプログラムを生成すればよい．しかし，インタプリタと同様のCプログラムを生成して，それをそのまま実行するのであれば，結果的にはNTOAMインタプリタを実行するのとあまり変わらないことになり，顕著な実行性能の向上は期待できない．むしろ，NTOAMのデータ構造からCの変数への変換のオーバヘッドのために性能が低下する可能性がある．そのようなオーバヘッドを上回る性能向上を実現するためには，NTOAMプログラムからCプログラムへの変換に際して効果的な最適化を行う必要がある．そこで変換の基本方針を以下のように定めた．

(1) β-Prologの最終出力であるNTOAMプログラムの代わりに，中間データである照合木を用いてCプログラムの生成を行う．
(2) 大域決定的な述語[7]のみをCプログラムへ変換する．

複数の命令語にまたがって最適化を行うためには，命令語間の依存関係を調べる必要がある．手続き型言語のコンパイラ[13]ではそのために制御フロー解析ならびにデータフロー解析を行う．Prologでは変数は再利用されないので，データフロー解析は不要である．また，β-Prologで用いている照合木は制御フローを表している．このようなことから上記の方針(1)で述べているように，本研究ではProlog-to-C変換系の入力を照合木とした．

一般に手続き型プログラムの動作は決定的である．それに対してPrologプログラムは非決定的に動作することがその特徴の1つである．しかし，Prologから手続き型言語Cへ変換する際には，この非決定性が

☆ membchk/2が他の照合木のテストノードから呼び出されている場合にシャローバックトラックとなる．

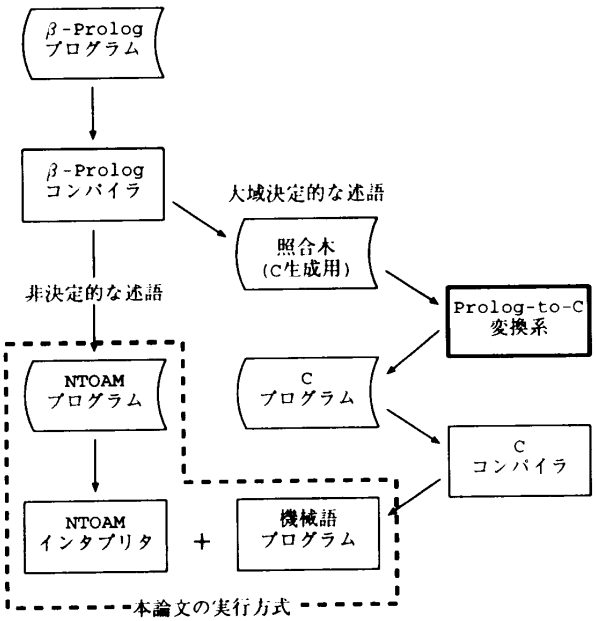

図4 Prologから C への変換と実行
Fig. 4 Translation of Prolog programs to C and execution.

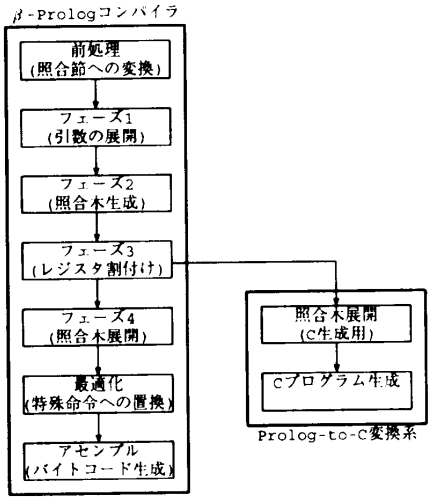

図5 照合木から C プログラムを生成する手順
Fig. 5 Procedure to generate C programs from matching trees.

困難な問題を引き起こすことになる．C プログラムにおいて非決定性を実現するためには，実行途中の状態をすべてスタックに保存し，逐次解釈的な実行を行わざるをえない．すなわち，非決定的な Prolog プログラムを C プログラムに変換したとしても，NTOAM インタプリタと同じ動作をすることになり，変換したことによる性能向上は期待できない．このようなことから，上記の方針(2)で述べたように，本研究では大域決定的な述語のみを変換の対象とした．

ここで，大域決定的な述語とは，その述語呼び出しが失敗した場合にバックトラックが起こらないような述語のことである．具体的には，その述語をヘッドとするすべての規則において，ボディに含まれるのは，ユニフィケーションや算術比較のようなインライン展開される述語，または他の大域決定的な述語呼び出しだけである．

Prolog では，決定的なアルゴリズムも，非決定的であるかのように記述することができるが，実践的なプログラミングとしては，カットの使用による非決定性の排除が推奨されている[14]．本研究ではそのような明らかに決定的な述語のみを，プログラマが変換の対象として指定する．ただし，変換された C プログラムは，非決定的な Prolog の述語から呼び出すことができるので，本研究の手法の適用範囲を決定的な Prolog プログラムに限定しているわけではない．図4に示すように，C へ変換されずに NTOAM によって解釈実行されるコードと C へ変換されたプログラムとをリンクして実行することができる．

3.2 変換の概要

β-Prolog コンパイラによって照合木を作成し，その照合木から C プログラムを生成する．その手順を**図5**に示す．

Prolog-to-C 変換系の照合木展開は，β-Prolog コンパイラのフェーズ4と基本的には同じであり，照合木のノードを一次元のコード列へ展開する．あとで C プログラムを生成するために以下のような処理を付加している．

- パラメータ，一時変数の管理
- アトム，述語名の管理
- deref の挿入

上の2つの処理は C において変数名や関数名を生成するために必要な処理である．

deref とは，変数のリンクをたどることである．Prolog でユニフィケーションが行われると，一方の変数から他方へリンクが張られる．したがって，変数の実体を得るためには deref を実行する必要がある．NTOAM の命令はすべて，それぞれの命令の機能として deref を行うようになっているので，β-Prolog コンパイラでは deref を扱っていない．しかし，命令ごとに deref を行うのは無駄が多いので，C プログラム生成に際しては，照合木上で実行順序を確認しながら，必要な箇所にだけ明示的に deref を挿入している．

照合木を展開すると，deref などの例外はあるが，基本的には NTOAM の命令系列が得られるので，各命令に対して，その命令の機能を実現する C のいくつかの文を生成する．ただし，Prolog の述語呼び出しを C の関数呼び出しに変換するためには少し工夫が必要

である．

β-Prologでは（Prologの）述語からCの関数を呼び出す場合，関数へは制御だけが渡され，引数は述語呼び出しの場合と同じように（Prologの）ヒープに格納されている．しかし，Cの関数相互に呼び出しを行うときには，通常の関数引数として渡す方がはるかに効率が良い．そこで本研究のProlog-to-C変換系では，Cへ変換されるそれぞれの述語に対して，要求されている処理を実現する関数とは別に，インタフェース関数を用意することにした．これによってPrologからの呼び出しではインタフェース関数を経由して本体の関数を呼び出すが，その本体の関数から別の（Cへ変換された）述語や自分自身を呼び出すときには，インタフェース関数を経由することなく，通常のCの関数呼び出しで処理することができる．

3.3 最適化

Prologプログラムのコンパイルに対する最適化技法[15),16)]を本研究の変換にも適用することができる．具体的には，本研究では，NTOAMの各命令をCへ変換する際に，以下に述べるような最適化を行った．

- derefの削除，移動
- 算術演算の最適化
- レジスタ変数の利用

3.2節で述べたように，NTOAMの各命令と切り離してderefを実行することによって，derefを減らしている．また，Prologプログラムにおいて尾部再帰が書かれていると，NTOAMプログラムではループとして実現されるが，ループ中で不変なオペランドに対するderefをループ外に移動させることによって，derefの実行回数を減らすことができる．

NTOAMの算術命令は2項演算であるので，一般の数式は一時変数を導入して演算が行われる．さらにNTOAMでは，数値データはタグ付きで表示されるので，演算のたびにオペランドのタグをはずし，演算結果にタグを付けるという操作を行うことになる．しかし，Cでは任意の数式を扱うことができるので，連続する算術命令をまとめて一時変数とそのタグ操作を削除することができる．

Cではレジスタ変数を宣言することによってより高速な処理が期待できる．しかし，一般に利用可能なレジスタ数はあまり多くないので，利用頻度の高いものをレジスタ変数としなければならない．そこで照合木を展開するときに，変数の参照回数を記録し，そのデータに基づいてレジスタ変数を決定している．

4. 評価

本研究で提案したCへの変換手法を評価するためにいくつかの実験を行い，実行性能ならびにプログラムの理解性の評価を行った．

4.1 実行性能の評価

いくつかの例題プログラムについて，他のProlog処理系と実行速度の比較を行った．

比較の対象とした処理系は，β-Prolog ver 1.0のNTOAMバイトコードへのコンパイラ，SICStus Prolog ver 2.1.6[17)]のバイトコードコンパイラおよびネイティブコードコンパイラ，そしてKL1からCへの変換を行うKLIC[18)]である．

例題プログラムは，小規模なものとしてmembchk（要素数10,000），nreverse（要素数500），quicksort（要素数600），8-queen（全解探索），大規模なものとして，β-Prologコンパイラ，本研究で作成したProlog-to-C変換系を用いた．コンパイラならびに変換系への入力データとしてはβ-Prologコンパイラのソースプログラム☆を与えた．

同じPrologソースプログラム（とその変換結果）を同じ環境（マシン，OS）で実行した結果を**表1**に示す．かっこ内の数値は本手法による実行時間を1.0としたときの比率である．例題プログラムのうちで8-queenは非決定性を含んでいるので，Cへ変換できたのは全体の55%であった．β-Prologコンパイラならびに本研究のProlog-to-C変換系にも一部非決定性が存在した．本実験では非決定的な部分は元のままNTOAM上で実行しており，表中の実行時間には，そのようなNTOAM上での実行時間も含んでいる．

この表に示したすべての場合において本手法による変換の結果がすぐれている．8-queenの例からもわかるように，非決定性を含んでいるような場合でも，決定的な述語を抽出してCへ変換することによって，かなりの高速化を達成することができる．

4.2 理解性の評価

プログラムの理解性を評価するために，McCabeの閉路複雑度やHalsteadのソフトウェア科学など，多くの尺度が提案されているが[19)]，ソースコードの行数以外の尺度をPrologプログラムに適用することはできない．そこで，1つの目安として，4.1節で用いた例題のうちで，Cへの変換率が100%であった3つの例題について，元のPrologプログラム，自動変換されたCプログラムの行数，人手で作成したCプログ

☆ β-Prolog自身で記述されている．

表1 実行時間の比較
Table 1 Comparison of execution time.

program	β-Prolog		SICStus Prolog		KLIC
	C	NTOAM	バイトコード	ネイティブコード	C
membchk	130	500 (3.85)	930 (7.15)	400 (3.08)	210 (1.62)
nreverse	190	710 (3.74)	650 (3.42)	230 (1.21)	390 (2.05)
quicksort	30	80 (2.67)	190 (6.33)	110 (3.67)	40 (1.33)
8-queen	160	400 (2.50)	630 (3.94)	220 (1.38)	-
β-Prolog コンパイラ	110370	227170 (2.06)	553610 (5.02)	282711 (2.56)	-
Prolog-to-C 変換系	40390	74050 (1.81)	157260 (3.84)	90700 (2.22)	-

(Sparc Station 2 で計測，単位は ms．かっこ内の数値は本手法による実行時間を 1.0 としたときの比率)

表2 プログラム行数の比較
Table 2 Comparison of lines of program.

program	Prolog	自動変換		人手で作成	
		C	C/Prolog	C	C/Prolog
membchk	3	26	8.7	26	8.7
nreverse	8	87	10.9	18	2.3
quicksort	13	112	8.6	48	3.7

ラムの行数を計数した．その結果を**表2** に示す．

表2 で自動変換された C プログラムの行数をみると，元の Prolog プログラムに対して 8～10 倍となっている．大規模な例題では C への完全な変換が行えないので正確な比較はできないが，本研究の Prolog-to-C 変換系が Prolog で約 2500 行であるのに対して，それを C へ変換した結果は約 22000 行であり，約 9 倍の大きさになっている．一方，人手で作成した結果は 2～8 倍となっており，人手で作成したからといって必ずしも小さくなるわけではないことを示している．いずれにしても，C プログラムは，元の Prolog プログラムよりも数倍以上のオーダで長くなり，それだけ理解が困難になると考えられる．

理解性についてより具体的に評価するために，membchk の Prolog プログラム（図1）とそれを人手で C へ変換したプログラム（**図6**）を比較する．図1 のプログラムが Prolog プログラマには一目瞭然であるのに対して，図6 のプログラムが何をしているのかを理解するためには，ソースプログラムをじっくりと検討しなければならない．さらにこのプログラムが間違いないものであることを確信するためには，データ構造の定義など，その他の文書を参照する必要がある．すべての問題に対して C 言語が不利であるわけではないが，特に Prolog で簡明に表現できるプログラムに関しては，それを C へ変換すると，理解性は低下すると考えられる．

人手で C のプログラムを最適化し，より小さくて

```
int
membchk(struct Atom x, struct Atom *listP)
{
  while( listP->type != NULL ) {
    switch(x.type) {
    case CHAR:
      if( x.value.c == listP->value.c )
        return 1;
      break;
    case INT:
      if( x.value.i == listP->value.i )
        return 1;
      break;
    case FLOAT:
      if( x.value.f == listP->value.f )
        return 1;
      break;
    case STR:
      if( !strcmp(x.value.s, listP->value.s) )
        return 1;
      break;
    }
    ++listP;
  }
  return 0;
}
```

図6 人手で作成した C プログラム
Fig. 6 Manually coded C program.

高速なプログラムを開発できたとしても，そのためには膨大な開発・保守コストが必要になる．それに対して，本研究で提案した C への自動変換のアプローチでは，開発ならびに保守はコンパクトで理解が容易な Prolog プログラム上で行い，実行は C プログラムで行うことにより，生産性や保守性を損なうことなく処理の高速化を実現することができる．

5. 関連研究

本研究と同じ WAM に基づく C への変換手法を用いている文献12) では，SICStus Prolog のネイティブコードコンパイラとの比較が行われていて，nreverse

で約60%，8-queenで約40%遅い結果が出ている．それに対して，本研究の手法は4.1節に示したように，SICStus Prologを上回る性能を得ることができた．

文献20)では，継続（continuation）をC関数の引数として与えることによってPrologの処理を実現する手法を提案しているが，ベンチマークテストなどの評価はなされていない．この手法では大きなスタック空間を使用するため，現実の大きな問題を解くのは困難であると著者らは考えている．

6. おわりに

本論文では決定的なPrologプログラムをCへ変換して高速に実行する手法を提案した．この手法によって抽象機械による実行と比較して2～4倍の速度向上を実現することができた．

知識処理の記述はCのような手続き型言語よりもPrologで記述する方が生産性や保守性にすぐれているが，性能上の問題でCなどで記述されることも少なくない．本論文で提案した手法を用いることによって，Prologの生産性や保守性を損なうことなく性能向上を実現することができる．

近年ハードウェアの変化が著しく，それにともなってOSなどのソフトウェア環境も大きく変化することが稀ではなく，ソフトウェアの移植性が重要な課題の1つとなっている．β-Prolog処理系および本研究のProlog-to-C変換系はいずれも標準的なC言語をベースとしており，特定のハードウェアやソフトウェアには依存していないので，移植性はきわめて高い．

本論文の手法は，非決定性を含む任意のPrologプログラムに適用できる．すでに，約1万行のPrologで記述されたβ-Prologコンパイラに適用しており，実用規模のプログラムに適用可能であることを確かめている．現在，別のProlog処理系で開発された設計計算支援システムDSP[5]のβ-Prologへの移植を進めており，本手法による高速化を適用する予定である．

謝辞 本論文に対して有益な助言をいただいた本学橋本正明教授ならびに梅田政信氏に感謝します．

参考文献

1) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: *Object-Oriented Modeling and Design*, Prentice Hall (1991). 羽生田栄一（監訳）：オブジェクト指向方法論—モデル化と設計，トッパン (1992).
2) Bell, R.: Choosing Tools for Analysis and Design, *IEEE Software*, Vol.11, No.3, pp.121–125 (1994).
3) Prieto-Diaz, R.: Domain Analysis for Reusability, *Proc. IEEE Computer Software and Application Conference*, pp.23–29 (1987).
4) 田村恭久，伊藤　潔，杵嶋修三：ドメイン分析・モデリング技術の現状と課題，情報処理，Vol.35, No.5, pp.952–961 (1994).
5) 梅田政信，長澤　勲，樋口達治，永田良人：設計計算のプログラム書法，電子情報通信学会技術研究報告（人工知能と知識処理），Vol.91, No.315, pp.25–32 (1991).
6) 廣田豊彦，橋本正明，長澤　勲：応用ドメインに特化した概念モデル記述言語に関する一考察，情報処理学会論文誌，Vol.36, No.5, pp.1151–1162 (1995).
7) Zhou, N.-F.: Global Optimizations in a Prolog Compiler for the TOAM, *J. Logic Programming*, Vol.15, No.4, pp.275–294 (1993).
8) Zhou, N.-F.: An Extended Prolog with Boolean Tables for Combinatorial Search, *Proc. 5th IEEE Int. Conf. on Tools with Artificial Intelligence*, pp.312–319 (1993).
9) Zhou, N.-F.: On the Scheme of Passing Arguments in Stack Frames, *Proc. 11th Int. Conf. on Logic Programming*, pp.159–174 (1994).
10) Ait-Kaci, H.: *Warren's Abstruct Machine: A Tutorial Reconstruction*, The MIT Press (1991).
11) Warren, D.H.D.: Implemeting Prolog-Compiling Predicate Logic Programs, Technical Report 39 and 40, Dept. of Artificial Intelligence, Univ. of Edinburgh (1977).
12) Levi, M.R. and Horspool, R.N.: Translating Prolog to C: WAM-based approach, *Proc. Second Compulog Network Area Meeting on Program Languages* (1993).
13) 佐々政孝：プログラミング言語処理系，岩波書店 (1989).
14) Sterling, L. and Shapiro, E.: *The Art of Prolog*, The MIT Pess (1986). 松田利夫（訳）：Prologの技芸，共立出版 (1988).
15) Taylor, A.: Removal of Dereferencing and Trailing in Prolog Compilation, *Proc. 6th Int. Conf. Logic Programming*, pp.49–60 (1989).
16) Debray, S.K.: A simple code improvement scheme for Prolog, *J. Logic Programming*, Vol.13, No.1, pp.57–88 (1992).
17) Swedish Institute of Computer Science: *SICStus Prolog User's Manual* (1991).
18) Chikayama, T., Fujise, T. and Sekita, D.: A protable and efficient implementation of KL1, *Proc. 6th Int. Symp. Programming Language Implementation and Logic Programming, PLILP '94*, pp.25–39 (1994).
19) McClure, C.: *The Three Rs of Software Au-*

tomation: Re-engineering, Repository, Reusability, Prentice-Hall (1992). ベスト CASE 研究グループ (訳): ソフトウェア開発と保守の戦略—リエンジニアリング・リポジトリ・再利用—, 共立出版 (1993).
20) Lock, H.C.R.: Issues in the implementation of Prolog, and their optimization, *Microprocessing and Microprogramming*, Vol.32, No.1-5, pp.505–514 (1991).

(平成 7 年 7 月 19 日受付)
(平成 8 年 3 月 12 日採録)

片峯 恵一 (正会員)

1992 年九州工業大学情報工学部機械システム工学科卒業, 1994 年同大学院情報工学研究科修士課程修了, 同年九州工業大学情報工学部助手, 現在に至る. 論理プログラミングおよびソフトウェア開発支援環境の研究に従事.

廣田 豊彦 (正会員)

1954 年生. 1976 年京都大学工学部電気工学第二学科卒業. 1981 年同大学院工学研究科博士後期課程研究指導認定退学. 工学博士. 1981 年京都大学情報処理教育センター助手. 1988 年九州工業大学情報科学センター助教授. 1989 年同大学情報工学部知能情報工学科助教授, 現在に至る. プログラム開発環境, プログラムテスト支援, CAD システムなどの研究に従事. 著書「C プログラミングの基礎」(培風館) ほか. 日本ソフトウェア科学会, 人工知能学会各会員.

周 能法 (正会員)

1984 年中国南京大学計算機科学系卒業, 1991 年九州大学情報工学研究科大学院博士課程修了. 工学博士. 同年, 九州工業大学情報工学部講師. 1993 年同学部助教授, 現在に至る. 論理プログラミングおよび並列処理の研究に従事. 日本ソフトウエア科学会, 人工知能学会, IEEE, ALP 各学会会員.

長澤 勲 (正会員)

1944 年生. 1967 年九州大学工学部電子工学科卒業. 1972 年同大学院工学研究科博士課程単位取得退学. 同年九州大学中央計数施設講師. 現在, 九州工業大学情報工学部教授 (機械システム工学科). 工学博士. 知識情報処理の立場から CAD/CAM, ロボット, 医療システム等の研究開発に従事. 人工知能学会, 日本建築学会, 精密工学会, 電子情報通信学会, 日本機械学会, 日本設計工学会, 日本ロボット学会各会員.