TR-586

知識ベース指向の並列推論処理システム

北上　始，横田　治夫（富士通）

August, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Knowledge-Base-Oriented Parallel Inference System

Hajime Kitakami and Haruo Yokota
Fujitsu Limited
1015 Kamikodanaka, Nakahara-Ku, Kawasaki 211, JAPAN

[Abstract]

It is well known that we need a knowledge-base-oriented parallel inference system which would support human problem solving on computer systems. Since today's systems tend to result in explosive combinatorial growth and large-scale knowledge bases, we need to speed up the system.

This paper describes a parallel programming method using a meta-programming technique based on the parallel logic programming language GHC. Since it does not include OR-parallel function which can realize knowledge retrieval system, we select the other method simpler than a method for representing knowledge bases as perpetual process. We represent data in the knowledge base using the $-variable, which is a specific constant, to distinguish it from any other GHC variable in GHC's interpretation system. Since we found no advantage in implementing $-variable processing in GHC, we wrote it in C. The system includes a knowledge retrieval system which can support a large-scale knowledge base and interface between it and GHC's interpretation system.

We present the performance of the parallel constraint logic programming system, including the kernel system, used for knowledge-base-oriented parallel inference.

知識ベース指向の並列推論処理システム

北上　始，横田治夫

富士通株式会社

〒211 神奈川県川崎市中原区上小田中1015

【概要】

人間の知的問題解決を計算機上で実現するためには，知識ベース管理機構と推論機構をあわせもつ知識ベース指向の推論処理システムが必要であると言われているが，知識ベースの大規模化や推論処理における組み合わせ爆発の傾向が強くなるにつれ，システムの高速性が要求される。

本論文では，このようなシステムの高速性に答えるために，これを並列化する方法について提案する。基本的な方式は，メタプログラミング の考え方を基礎にしている。本システムを実現するために利用した並列論理型言語GHC は，AND 並列処理の機能を持つが，並列に知識ベース検索を行うようなOR並列処理の機能を持っていない。本システムでは，知識ベースをGHC の永久プロセスで実現する方法よりも簡単にプログラミング ができる方法を採用する。即ち，並列推論処理を実現するために利用したGHC の変数と，知識ベースを表現するための変数を区別するような，$ 変数を知識ベース側に導入する。これにより，$変数を扱う単一化処理や代入処理を実現する。また，これらの$変数に関連する処理は，GHC で実現しても効果がないことから，これらを逐次言語Cで実現する。さらに，大規模な知識ベースに対処するために，ハッシュやトライ で構造化されている知識ベースに対して，GHC からアクセス可能にする。

最後に，本システムの並列性能を評価するために，本システムを中核とする制約論理プログラミングシステム を作成し，その測定を行ったので，その結果について報告する。なお，測定においては，共有メモリー型のマルチマイクロプロセッサ Symmetry S81 を使用した。

【　目次　】

ページ

1．はじめに　— 3 —

2．メタプログラミング　— 5

3．GHC による並列化　— 7 —

3.1　$ 変数の導入

3.2　並列プログラミング

3.3　カットシンボル の処理

4．大規模な知識ベース検索　13

5．システムの評価　—15 —

5.1　静電界の問題

5.2　測定結果

6．おわりに　—19 —

【参考文献】　—20 —

1．はじめに

人間の知的問題解決を計算機上で実現するためには，知識ベース管理機構と推論機構をあわせもつようなシステムが必要と言われている〔Fuchi '78 〕。このような知識情報処理を目指したシステムは，ｵﾌﾞｼﾞｪｸﾄ指向や制約指向のﾌﾟﾛｸﾞﾗﾐﾝｸﾞﾊﾟﾗﾀﾞｲﾑを吸収しながら，医療分野・電気機械システムの診断や分析をはじめ，電力系統の設備計画・生産計画・輸送計画等，さらには，電気機械システム等の設計や自然現象等のｼﾐｭﾚｰｼｮﾝにも利用されるようになってきている。

しかし，知識ベースの大規模化や推論処理における組み合わせ爆発の傾向が強くなるにつれ，システムの高速化が，益々，強く要求されるようになってきている。そのため，並列推論マシンPIM 〔Uchida '88, Hattori '89 〕の出現が大変期待されている。特に，遺伝子情報解析〔Naito '90 〕・輸送計画〔Fukumura '88〕・LSI の設計〔Hayashi '73 〕・粒子系のｼﾐｭﾚｰｼｮﾝ〔Sato '90〕等では，数時間を遙かに超えるような事例も数多く見受けられるようになってきている。並列推論マシンの利用は，記号処理のサイドに立って，上記の問題を高速に解くための有効な一つの研究アプローチであると考えている。

本論文では，並列推論マシンPIM の実験機として，共有ﾒﾓﾘｰ型のﾏﾙﾁﾏｲｸﾛﾌﾟﾛｾｯｻSymmetry S81 を使用し，知識ベース検索機構を含む並列推論処理システムをﾒﾀﾌﾟﾛｸﾞﾗﾐﾝｸﾞ〔Bowen & Kowalski '81, Kitakami '84〕の考え方を基礎にして，試作・評価したので，その結果について述べる。

本システムの基本動作は，人間の知識が登録されている知識ベースに対し，①並列推論を進めるために必要なデータをOR並列で検索した後，②並列推論の中でそのデータをAND並列で分析・加工する処理であると，見做している。しかしながら，従来の記号処理向きの並列ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ 言語では、これらの二つのタイプの並列処理の研究は，別々に進められていた。例えば，OR並列に力点を置く研究には，PRISM 〔Kasif '83 〕，P-Prolog〔Yang '87〕などがあり，AND 並列に力点を置く研究には，GHC 〔Ueda '85, Ozawa '89 〕，Parlog〔Clark '84 〕，Concurrent Prolog 〔Shapiro '84 〕などがある。本システムを実現するには，OR並列とAND 並列の両方を記述する機能が必要であることから，何れの並列ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ 言語でも難しい。OR並列処理の部分をAND 並列処理に変換する研究〔Takeuchi '87〕も精力的に進められてはいるが，完全に変換する方法は未だ得られていない。

本論文では，並列推論マシンPIM の核言語として，GHC が選択されているという事情により，AND 並列言語GHC にOR並列の知識ベース検索機構を導入することにより知識ベース

指向の並列推論処理を行う方法について述べる。OR並列の知識ベース検索機構を，AND 並列言語GHC に導入するために，単一代入制限をもつGHC の変数と知識ベースを表現する変数を完全に区別する方法を提案する。本システムは，拡張性に富むメタプログラミング の方法を基礎にして実現されているので，カットシンボル の処理が自然に実現できることも示す。また，GHC 処理における無駄なメモリー消費を回避するために，知識ベースに対する変数処理をGHC で作成することを避け，逐次言語C で実現している。さらに，大規模な知識ベースの高速アクセスを実現するために，知識ベース検索機構〔Yokota '89〕をGHC から並列にアクセスする方法を示す。具体的には，GHC プロセスと交信可能な知識ベースの検索プロセスを共有メモリー上に実現し，お互いのプロセスのインタフェース を取ることができる。検索プロセスは，知識ベースへの検索コマンド毎に用意される。

最後に，本システムを評価するために，本システムを中核とした制約論理プログラミングシステム を作成し，その測定結果について示す。測定では，静電界の問題〔Heintze '87 〕の例題を扱っている。一般に，制約論理プログラミングシステム 〔Lassez '87〕は，①知識ベースから制約式を抽出する処理と，②制約式を評価する処理とに分けられるが，ここでは，前者を，本システムにより実現し，後者を，連立一次方程式に帰着されるような制約式に限定し，これをGHC によるAND 並列処理により実現している。測定の結果，並列知識ベース検索に1台のPEを割付け，GHC 処理に9台のPEを割りつけることにより，GHC 処理に1台のPEを割りつけた場合と比較し，約9倍の並列性能が得られている。

以上により，マルチマイクロプロセッサ Symmetry S81 上で，効率的に並列動作する知識ベース指向の並列推論処理システムが試作されたことを示す。

なお，本研究は，第五世代コンピュータプロジェクトの研究の一環として，財）新世代コンピュータ技術開発機構からの再委託研究として行われたものである。

2．メタプログラミング による推論

並列に動作する本システムの基本的な実現方式は，論理型プログラミング 言語Prologのメタプログラミング の考え方をヒントにしている。従って，ここでは，並列処理との関わりに触れながらメタプログラミング による推論処理方法について簡単に説明する。

通常のプログラミング では，メタレベル の言語処理系の上で，オブジェクトレベル のプログラミング をしていたが，メタプログラミング は，オブジェクトレベル のプログラム をメタレベル で解釈するようなプログラミング 形式を指す。これにより，メタプログラミング では，メタレベル のプログラミング が自由にできるようになるので，言語処理系に直接手を加えて修正せずとも，言語処理系の機能を拡張したプログラミング が容易に可能になる。

知識情報処理システムでは，図１の例で示されるような知識ベースがオブジェクトレベル として位置づけられ，図２の例で示されるような推論処理がメタレベル として位置づけられる。

```
ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).

parent(f1,f2).
parent(f2,f3).
parent(f3,f4).
```

図１．知識ベースの例

```
solve( true):- !.
solve((P,Q)):- solve(P),solve(Q).
solve( P):- clause(P,Q),solve(Q).
```

図２．推論処理の例

図１の知識ベースは家系図を表すデータであり，１行目及び２行目は，先祖の概念を表現するルールである。先祖を表す項 "ancestor(X,Y)" は，ある人 "X" の先祖が "Y" であることを指す。先祖は両親の１人を表す述語 "parent(X,Y)" に基づいて再帰的に定義されるが，両親の１人を表す述語は，ある人 "X" の両親の１人が "Y" であることを指す。

図２のメタプログラミング により実現された推論処理プログラム は，最も簡単なPrologインタプリタであり，探索木上で１つの解答を探索するプログラム である。問題 "?- solve(Goals)." が与えられると，その中のゴール列 "Goals" は，２行目のルールにより単一のゴールに分解される。このゴールの分析を進めるために，３番目のルールの中で，知識ベース検索が行われ，検

索結果得られたルールの条件部の分析がさらに進められる。この分析は，条件部の真偽が付くまで行われる。ゴールが，１行目の停止条件を満たす時，真であり，３番目のルールの中で，知識ベース検索の述語“clause(P,Q)”　の答えが得られない時，偽となる。偽となる場合は，他の可能性を調べるために，探索木上で推論処理のﾊﾞｯｸﾄﾗｯｸ が行われる。

図１及び図２からも自明なように，ﾒﾀﾌﾟﾛｸﾞﾗﾐﾝｸﾞ では，ﾃﾞｰﾀ としての知識ベースもﾌﾟﾛｸﾞﾗﾑとしての推論処理も区別しないので，先祖を定義するために利用されている変数“X”，“Y”，“Z”も，推論処理のﾌﾟﾛｸﾞﾗﾐﾝｸﾞ で利用されている変数“P”，“Q”も，通常のPrologﾌﾟﾛｸﾞﾗﾐﾝｸﾞ で利用される変数と同じタイプの変数として利用される。

さて，図２で示される推論処理により取敢えず解答を１つ求めることができるが，この処理は，探索木上のある一本のパスを辿る処理なので，この部分の効率的な並列処理が難しい。例えば，“?- solve(ancestor(X,Y)).”を実行すると，“X= f1”，“Y= f2”が計算されるが，この処理の部分を，AND ｽﾄﾘｰﾑ 並列で並列処理ができても，大きな並列性能を引き出すことは，難しい。従って，この推論処理を沢山の経路を同時に辿るように拡張し，全解を求めるための推論処理とすると，効果的な並列性能が期待出来そうである。即ち，この拡張された推論処理では，複数の解答を同時に探すために，探索木上を並列に探索することができる。全解探索用の推論処理は，種々の応用を持つ。特に，後述するような制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ や知識ベースの矛盾や冗長性等を取り除くためにも利用される。
次章以降では，以上のﾒﾀﾌﾟﾛｸﾞﾗﾐﾝｸﾞ に基づいて拡張した全解探索用の推論処理を並列化する方法について述べるが，その中では，知識ベース検索処理の実現方法が重要になる。そこで，知識ベース検索処理を中心に，全解を求める推論処理の手続きを整理すると，次のようになる。

(1) 知識ベース検索

ゴール列の中の各ゴールについて，ゴールと単一化が可能なホーン節を知識ベースから検索する。知識ベース検索の結果，答えが見つからない時，処理(4) を行い，答えが見つかった場合は，一つ答えが見つかる毎に，処理(2) を行い，答えが無くなったとき，処理(4) を行う。

(2) 変数名の付替え

知識ベースで定義されているホーン節と知識ベースから検索されたホーン節は異なるホーン節として扱われるので，これらのホーン節にまたがって変数名の共有が許されない。従って，検索されたホーン節内の変数に新しい変数名を付け，(3)の処理へ進む。

(3) ホーン節の分析

検索されたホーン節が，条件部を持っていれば，条件部を新たなゴール列とし，(1)の処理へ進む。条件部がなければ，常に，ホーン節が正しいので，推論結果の答えを出力し，次の答えを計算するために，(4)の処理へ進む。

(4) ﾊﾞｯｸﾄﾗｯｸ 処理

ゴール変数にバインドされているデータを解除し，探索木上で共通の親ノードを持つ他のゴールを(1) で探すことにより，ﾊﾞｯｸﾄﾗｯｸ を行う。他のゴールが無ければ，さらに上位の親ノードをゴールとして，(4)の処理を繰り返す。さらに，上位の親ノードが無い場合は，処理を停止する。

3．GHC による並列化

前章で説明したように，全解を求めるための推論処理は，(1) 知識ベース検索，(2)変数名の付替え，(3)ホーン節の分析，(4)ﾊﾞｯｸﾄﾗｯｸ 処理，として表される。前述したように，これらの処理の中で，(4)の処理を並列化すると，ﾊﾞｯｸﾄﾗｯｸ処理は，探索木上で，複数の解を並列に探索する処理に置き換えられる。

しかしながら，これらを並列言語GHC で記述する場合，(3) はﾒﾀﾌﾟﾛｸﾞﾗﾐﾝｸﾞ の考え方を基礎にして実現できるが，GHC はOR並列処理の機能がないので，(1) の処理を実現するのが難しい。即ち，GHC では，変数に多重値を取ることが許されていないので，1 つの変数で知識ベース検索により得られる複数の解答を受け取ることが難しい。また，(2) の処理を行う為には，変数と定数を区別する機能がなければならないが，GHCには備えられていない。さらに，(4)の並列探索では，複数のパスを並列に探索するために，パス毎に，一つのゴールを用意しなければならない。そのためには，変数を含むゴールをコピーする必要があるが，GHC の単一代入制限により，それが難しい。

3.1 $ 変数の導入

本システムでは，前述のGHC 変数に関連した問題点を解決するために，知識ベースをGHC の永久ﾌﾟﾛｾｽで実現するよりも簡単にﾌﾟﾛｸﾞﾗﾐﾝｸﾞ が可能な方法を採用する。即ち，GHC の変数と知識ベースのデータに含まれる変数を，分離し，知識ベースのデータに含まれる変数を特殊な定数で表現する。以後，この特殊な変数を$ 変数〔Kitakami '88〕，〔Kitakami

'89] と呼ぶことにする。

図３に＄変数を含む知識ベースの表現例を示す。GHC プログラミング において， ＄変数名の区別は，$(1),$(2)・・・で行っているが，以下では，簡単のため，これらを$1,$2,$3・・・で表現する。知識ベース検索の条件指定は，次のように表現される。

```
?- clause-stream( ancestor($10,$11), Out-Stream),
 Out_Stream= [ [(ancestor($12,$13):- parent($12,$13)), Binf1] ,
               [(ancestor($14,$15):- parent($14,$16),ancestor($16,$15))] ,Binf2] ]
```

"Binf1 " 及び "Binf2 " は，知識ベース検索に伴い単一化処理によって生じた＄ 変数のバインド情報である。

```
ancestor($1,$2):- parent($1,$2).
ancestor($1,$2):- parent($1,$3),ancestor($3,$2).

parent(f1,f2).
parent(f2,f3).
parent(f3,f4).
```

図３．知識ベースの例

図３に示すように，知識ベースのデータを，＄ 変数で表現すると，＄変数は，GHCにとって特殊な定数なので，単一化処理を，GHC処理系の機能を直接利用して，行うことが出来なくなる。そのために，＄ 変数を持つ項間の単一化命令及び＄変数を持つ項に対する代入命令を実現する必要がある。

(1) unify(Term1, Term2, NewTerm, OutputBinf).

"Term1 "（ 項）と "Term2 "（ 項）を単一化し，その結果を "NewTerm " に返す。"OutputBinf" には，単一化の結果得られた＄変数のバインディング 情報を返す。また，単一化処理が終わっても，"Term1 " と "Term2 " の＄変数は，書き換わることがない。例えば，ゴールとして，"unify(p(a,$1), p($2,b), Result, OutputBinf)" が与えられると，"Result = p(a,b) " 及び " OutputBinf = [b($1,b), b($2,a)] " が出力される。

(2) substitute(InputBinf, Term, Result, OutputBinf).

"InputBinf " に示される＄ 変数のバインディング 情報に基づき，"Term"（ 項）を書き換える。また，その結果を "Result" に返し，この処理により得られた＄変数のバインディング 情報を，"OutputBinf" に返す。例えば，ゴールとして，"substitute([b($1,b), b($2,a)] , p($1,$3), Result,Binf)" が与えられると，"Result = p(b,$3)" 及び " OutputBinf = [b($1,b)] " が出力される。

3.2 並列プログラミング

全解を求めるための並列推論処理は，前述の$変数を含む項に対する単一化処理や代入処理を利用し，メタプログラミング の考え方を基礎にすることにより実現できる。ここでは，メタプログラミング の考え方を使いどのように並列プログラミング を行うかについて説明する。

図2で示した推論処理は，バックトラック により解答を一つずつ求める処理であるが，本システムの並列推論処理は，複数の解答を並行に求める処理であるので，図2のバックトラック の中の知識ベース検索結果を一つずつ返す述語“clause(P,Q)”を拡張し，知識ベース検索により検索されるホーン節集合をストリーム として受け取るようにしなければならない。前節の“clause-stream”により受け取ったホーン節を順次分析していくような並列推論処理は，次のような並列プログラム〔Kitakami '90〕になる。

```
bagof__solve( [] , VL, Binf, His, Result):- true |
        BinfでHis の内容を書き換えたものをHis1とする,
        Result = [ [VL,Binf,His1] ] .
bagof__solve( [true] , VL, Binf, His, Result):- true |
        BinfでHis の内容を書き換えたものをHis1とする,
        Result = [ [VL,Binf,His1] ] .
bagof__solve( [ P| Q] , VL, Binf, His, Result):-  Q ≠ [] |
        bagof __solve( [ P] , VL, Binf, His, Result1),
        and __solve( Q, Binf, Result1, Result).
bagof__solve( [ P] , VL, Binf, His, Result):- otherwise |
        clause__stream( P, ClauseStream),
        or__solve( ClauseStream, VL, Binf, His, Result).
```

“bagof __solve”の第1引数には，解くべきゴール列が入力される。ここでは，ゴール列をリストで表現している。第2引数には，解かれたゴール列の出力変数がリスト入力される。第3引数には，推論中に$変数にバインドされた情報がリスト出力される。第4引数には，推論中に使用されたホーン節が使われた順にリスト出力される。第5引数には，推論結果として得られた複数の解答がストリーム として返される。

第1番目と2番目のプログラム は，図2の第1番目のプログラム に対応する処理である。上記の

第３番目と４番目のﾌﾟﾛｸﾞﾗﾑ は，各々，図２の第２番目と３番目のﾌﾟﾛｸﾞﾗﾑ に対応する。これらのﾌﾟﾛｸﾞﾗﾑ は，複数解を並行に探索する処理であるので，知識ベースから検索された複数のホーン節を次々と処理していくﾌﾟﾛｸﾞﾗﾐﾝｸﾞ になっている。特に，第３番目のﾌﾟﾛｸﾞﾗﾑ は，AND ゴール列を左から順に解いていくﾌﾟﾛｸﾞﾗﾑ である。この時，ANDゴール間を共有する$ 変数についても，先に処理されたゴールの$ 変数値を他のゴールに伝播させるために，第３番目のﾌﾟﾛｸﾞﾗﾑ で，変数 "Result1 " を "bagof __solve " から "and __solve " に渡している。第３番目のﾌﾟﾛｸﾞﾗﾑ の "and __solve " は，次のような並列ﾌﾟﾛｸﾞﾗﾑ によって実現される。

```
and __solve( Q, Binf, [ [ VL, BinfHis, His ] | R ], Result) :- wait(Q) |
        3.1 節の "substitute" により，ゴール列 "Q " に変数ﾊﾞｲﾝﾄﾞ情報 "BinfHis "
        を代入し，これにより更新されたゴール列を "Q1" として作成，
        bagof __solve( Q1, VL, BinfHis, His, Result1),
        and __solve( Q, Binf, R, Result2),
        merge( Result1, Result2, Result).
and __solve( Q, Binf, [], Result) :- wait(Q) | Result = [] .
```

"and __solve " の第１引き数には，解くべきゴール列が入力される。第２引き数には，推論中に＄変数にﾊﾞｲﾝﾄされた情報が入力される。第３引き数には，ゴール列 "Q " の直前に処理されたAND ゴールの処理結果が入力される。第４引き数には，ゴール列 "Q " の推論結果として得られた複数の解答がｽﾄﾘｰﾑ として返される。

第１番目のﾌﾟﾛｸﾞﾗﾑ は，ゴール列 "Q " の直前に処理されたAND ゴールの処理結果が１つ以上の解答を持つ時に，起動される。共有する＄変数へ変数値の伝播を行うために，直前に処理されたAND ゴールの処理結果を１つずつ "Q " に反映させ，それらのゴール列を解く。それらの解いた結果をマージし，マージした結果を "Result" に出力する。第２番目のﾌﾟﾛｸﾞﾗﾑ は，ゴール列 "Q " の直前に処理されたAND ゴールの処理結果が１つもない，即ち，[] のとき起動される。AND ゴール列の中のゴールが１つでも [] のとき，AND ゴール列の答えが，１つも得られないので，"Result" に [] が出力される。

ﾌﾟﾛｸﾞﾗﾑ “or__solve ” は，次のように実現される。

```
or__solve( [ [(Head:-Body).Inf] | ClauseList] , VL, Binf, His, Result):-true |
        3.1 節の “substitute” により，検索結果得られた$ 変数のﾊﾞｲﾝﾄﾞ情報Inf で
        変数ﾘｽﾄ VLを書き換え，NewVLを作成，
        append( Inf, Binf, NewBinf),
        append( His, [(Head:-Body)] , NewHis),
        bagof __solve( Body, NewVL, NewBinf, NewHis, Result1),
        or__solve( ClauseList, VL, Binf, His, Result2),
        merge( Result1, Result2, Result).
or__solve( [] , VL, Binf, His, Result):-true | Result = [] .
```

第１番目のﾌﾟﾛｸﾞﾗﾑ は，“clause__stream” によって検索された複数のデータを再帰的に１件ずつ処理するﾌﾟﾛｸﾞﾗﾑ である。“clause__stream” によって書き換えられた$ 変数は，$ 変数のﾊﾞｲﾝﾄﾞ情報Inf に返されているので，この情報を変数ﾘｽﾄ VLに反映させる。また，検索されたデータの履歴を取るために，データ “(Head:-Body)” を履歴情報 “His ” に追加する。データの条件部 “ Body ” は，“bagof __solve ” により分析が進められる。この結果 “ Result1” と “ ClauseList ” の分析結果 “ Result2” を，マージし，その結果を “Result” に出力する。第２番目のﾌﾟﾛｸﾞﾗﾑ は，検索結果データが１つも存在しなかった時か，検索された複数のデータの再帰的な処理が終了した時に，実行される。この時は，“Result” に [] が出力される。

3.3 カットシンボルの処理

前述したプログラミング スタイルをさらに押し進めると，カットシンボル の入った知識ベースに対する並列推論処理への拡張も容易に可能になる。

ここでは，カットシンボル は，簡単のため，１つのホーン節に一箇所だけ利用されるものと仮定し，条件節 "P, !, Q" を，以下のような "ifthen(P, Q) " で表現する。

```
children( $1, $2):-
      children( $1, [] , $2).
children( $1, $2, $3):-
      parent( father( $1, $4), children( $1, [$4 | $2] , $3)).
children( $1, $2, $2).
```

この知識ベースの例では，" children($1, $2)" に対する質問処理で，両親の中の1人 " $1 " に対する子供を全て見つけ，その結果をリストで " $2 " に返すことを期待している。この例で示されるような "ifthen(P, Q) " を処理するための並列プログラム は，前節の "bagof-solve " の第３番目と第４番目との間に，以下のようなプログラム を追加することによって実現される。

```
bagof __solve( [ifthen(P, Q) ] , VL, Binf, His, Result):-true |
     bagof __solve( P, VL, Binf, His, Result1),
     then__part( Q, Binf, Result1, Result).
```

" then-part" は，" P" の分析結果を使って，"ifthen(P, Q) " の " Q" を分析するプログラム であり，次のような並列プログラム として実現される。

```
then__part( Q, BindInf, [ Result1 | Result2 ] , Result):- true |
     and __solve( Q, Binf, [ Result1] , Result2),
     Result2 が [] ならば Result に [fail] を返し，
      [] 以外の場合はResultにResult2 を返す．
then__part( Q, BindInf, [] , Result):- true | Result = [] .
```

さらに，前節の“or-solve”のﾌﾟﾛｸﾞﾗﾑに，以下のようなﾌﾟﾛｸﾞﾗﾑを追加することによって実現される。

```
or__solve( [ [( Head:- [ifthen(Cond,Body) ] ),Inf ] |ClauseList] , VL,
          Binf, His, Result):-true |
      3.1 節の“substitute”により，検索結果得られた$ 変数のﾊﾞｲﾝﾄﾞ情報Inf で
      変数ﾘｽﾄ VLを書き換え,NewVLを作成,
      append( Inf, Binf, NewBinf),
      append( His, [( Head:- [ifthen(Cond,Body) ] ) ] , NewHis),
      bagof __solve( [ifthen(Cond,Body) ] , NewVL, NewBinf, NewHis, Result1),
      next__solve( ClauseList, VL, Binf, His, Result1, Result).
```

このﾌﾟﾛｸﾞﾗﾑ内の“bagof __solve ”により“ ifthen(Cond,Body)”が分析されると，“ next __olve”では，“Result1 ”が[] のとき，次の“ClauseList”を，“or__solve ”で分析し，それ以外のとき，“ClauseList”を捨て，枝刈りを行う。

4．大規模な知識ベース検索

これまでの説明では，知識ベース検索処理“clause-stream ”を含んだ並列推論処理“bagof-solve ”を全て並列言語GHC で実現していたが，このアプローチでは，$ 変数の単一化処理“unify ”及び代入処理“substitute”の$変数処理に関わる部分が低並列であるにも関わらずメモリーの消費量が大きい事などの問題がある。特に，大規模な知識ベース処理では，非常に大きな問題となる。また，GHC によるアプローチでは，永久プロセスを作って大規模な知識ベースのデータを効率的に管理・利用するのが難しい。

以上の点から，現在は，知識ベース検索処理“clause__stream”及び$ 変数の単一化処理“unify ”や代入処理“substitute”を，逐次言語のC言語により作成している。GHC からC言語で作成した単一化処理や代入処理を呼び出すことは，容易であるが，知識ベースのデータは複数のGHC プロセスで共有されなければならない。従って，このデータを管理する知識ベース検索処理RBU(Retrieval By Unification) 〔Yokota ’89〕と並列言語GH

C との間で，注意深くｲﾝﾀﾌｪｰｽ をとる必要がある。以下では，GHC とｲﾝﾀﾌｪｰｽ をもつ知識ベース検索を並列知識ベース検索と呼び，その実現方法を簡単に説明する。

図４に並列知識ベース検索処理部のシステム構成を示す。使用した並列マシンが，Sequent 社製の共有メモリー型ﾏﾙﾁﾏｸﾛﾌﾟﾛｾｯｻ であることから，UNIXベースの並列OS(DYNIX)上で試作されている。知識ベースのデータは，大規模なデータも扱えるようにするために，HashとTrieで構造化し，共有メモリー上に置かれている。共有メモリー上に置かれたデータは，複数のRBU から並列にアクセスされる。各RBU は，各々，UNIXの１プロセスに割りつけられる。この並列アクセスに対する排他制御は，項関係と呼ばれるデータの集合毎に行われ，データ検索かデータ更新かによって，共有モードと排他モードとを使い分けている。GHC からの知識ベース検索は，GHC プロセスとRBU プロセスとの間の連絡により行われる。

このシステムを試作し，検索性能を測定した結果，RBU に6 台のプロセッサ，GHC に10台のプロセッサを割り当てると，Quitus-Prolog の25倍の性能を得ることが分かている。また，GHC へ割り当てるプロセッサの台数を増やすことにより，良好な台数効果が得られることも分っている。

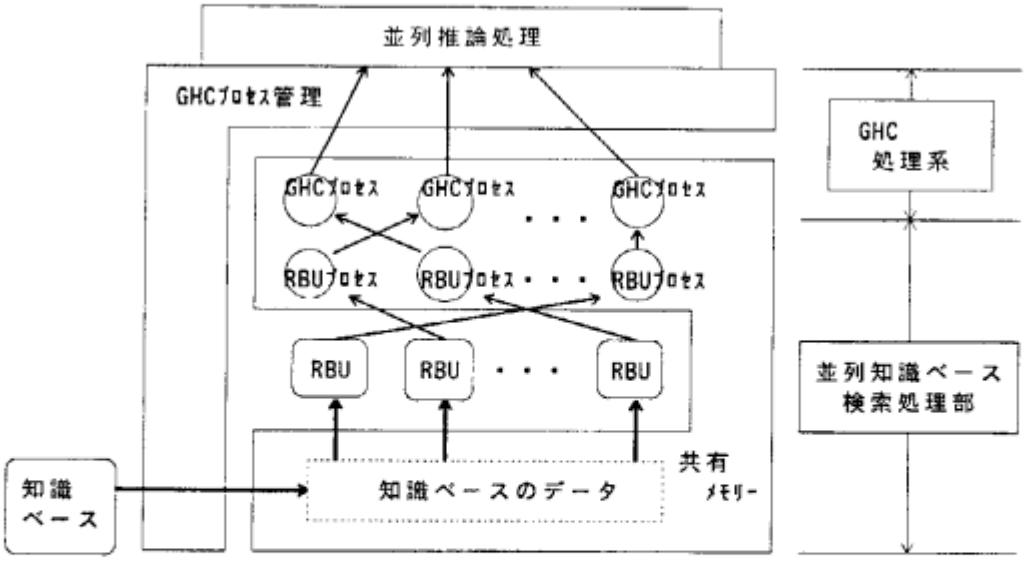


図４．システム構成

5．システムの評価

本システムは，種々の知識情報処理システムの中核として利用可能である。現在までに，図3に示されるようなﾃﾞｰﾀﾍﾞｰｽを含む問題[20]や“append”処理等のようなﾃﾞｰﾀﾍﾞｰｽを含まない問題について，良好な並列性能が得られることを確認している。ここでは，並列推論処理システムを中核とする制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑの評価を試みる。評価に当たっては，制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑの代表的な例題である静電界の問題を採用する。制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑは，推論中に，知識ベースから数式を抽出できる様に，3.2節の“bagof-solve”を拡張し，さらに制約処理系を付け加えることによって，容易に実現することができる。

従って，本章では，例題としての静電界の問題の説明，及び並列推論処理システムを中核とする制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑの測定結果の説明を行う。

5.1 静電界の問題

図5に示される二次元矩形領域に於いて，この領域内のある点の電位がある条件を満足するように，上端の電位を決定する問題について説明する。この領域はメッシュで分割されているが，各格子点の電位は，図6に示されるようなLiebmannの5点近似法で定められるものとする。

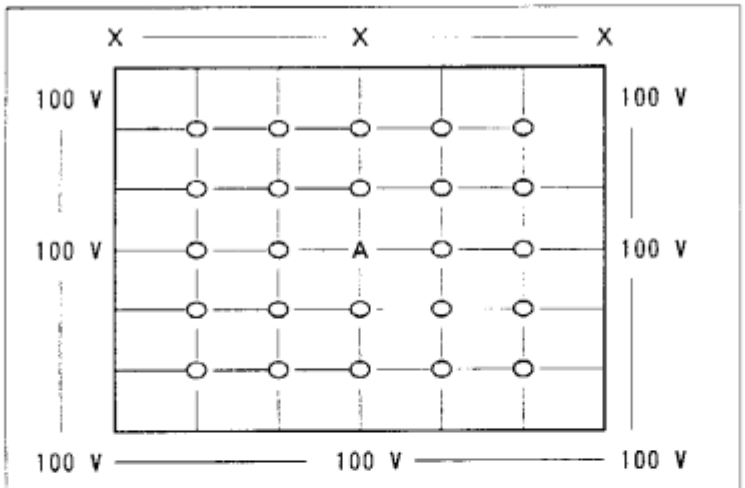
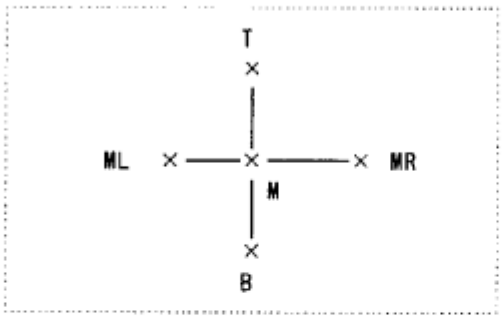


図5．二次元矩形領域

（　ただし，74.9 ≦ A ≦ 75.1　）



$$M = \frac{T + B + ML + MR}{4}$$

図６．Liebmannの５点近似法

この問題は，図７で表されるような知識ベースとなる。代数式の等号は，簡単のため，$ 変数の単一化命令“unify ”で表現することを避け，不等号と同じく，直接，算術記号“ =” を使っている。

```
design(  [$1 | $2] , $3, $4 ):-
     member( $3, [-1.0,1] ), laplace(  [$1 | $2] ), 74.9 ≦$4, $4≦ 75.1 .

member( $1,  [$1 | $2] ).
member( $1,  [$2 | $3] ):-  member( $1, $3).

laplace( [ $1, $2 ] ).
laplace( [ $1, $2, $3 | $4] ):-
     laplace-vec( $1, $2, $3), laplace(  [ $2,$3 | $4] ).

laplace __vec( [ $1, $2 ] ,  [ $3, $4 ] ,  [ $5, $6 ] ).
laplace __vec( [ $1,$2,$3 | $4] ,  [ $5,$6,$7 | $8] ,  [ $9,$10,$11 | $12 ] ):-
     $10 + $2 + $5 + $7 - 4 ×$6 = 0,
     laplace__vec( [ $2,$3 | $4] ,  [ $6,$7 | $8] ,  [ $10,$11 | $12 ] ).
```

図７．静電界の問題を解くための知識ベース

4.2 測定結果

前節の知識ベースに対して，次のような問い合わせをすると，

```
?- constraint__solve( design( [ [ $900,  $900,$900,$900,$900] ,
                                [  100,  $911,$912,$913,  100] ,
                                [  100,  $921,$999,$923,  100] ,
                                [  100,  $931,$932,$933,  100] ,
                                [  100,   100,  100,  100,  100] ] , $900,  $999)).
```

最初に，並列推論処理 " bagof__solve " が呼び出される．ここでは，図7の "member" の要素毎に，連立一次方程式を作成する。次に，制約処理系が呼び出され，この方程式が制約処理系に渡される。制約処理系では，これらを行列に変換し，行列計算を行う。行列計算は，図8に示すようなGHC のAND 並列ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ により容易に実現可能である。

図8では、4×4行列の例を図示している。各行列要素をGHC ﾌﾟﾛｾｽの単位(以下では並列処理単位と呼ぶ)とし、対角要素の上から下へと順に掃き出し法のアルゴリスムを適用して行く。

例えば，①の対角要素を起点とする計算は以下のように行う。

(1) 対角要素は，水平方向要素及び垂直方向要素へメッセージを送信する。

(2) 水平方向要素及び垂直方向要素がメッセージを受けた取ったならば、各要素は，それぞれ，垂直方向要素及び水平方向要素にメッセージを送信する。

(3) これにより，垂直方向及び水平方向からのメッセージを受けた取った各要素は，ただちに，掃き出し法に基づく計算を行う。

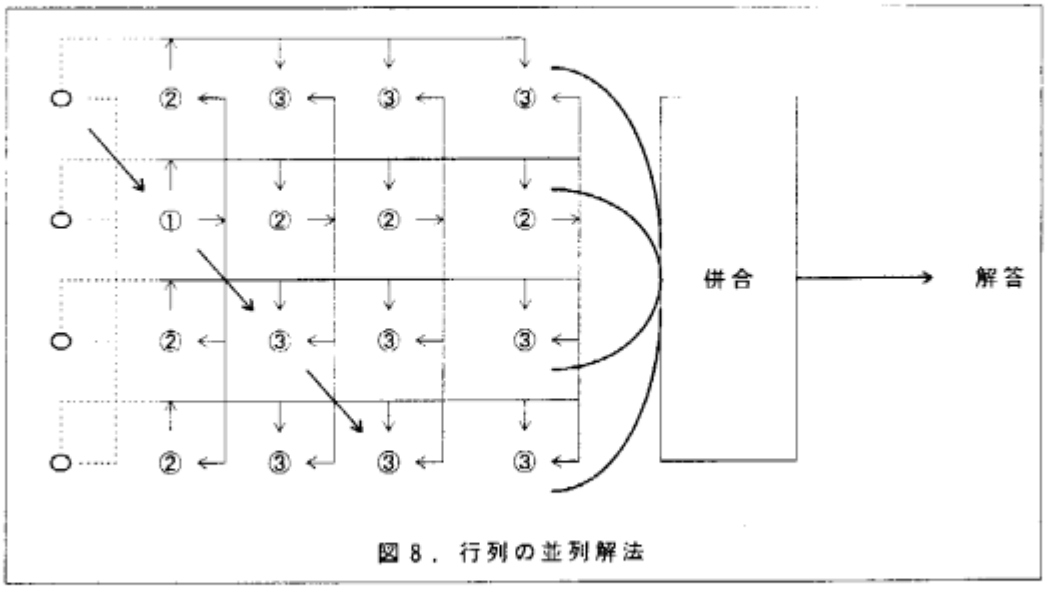


図8．行列の並列解法

並列行列解法の部分の並列性能は，GHC が 8 台のとき，1 台に比べて 6 倍近い並列性能を得ている。

制約処理を結合した並列推論処理システムでは，RBU に 1 台又は 6 台のプロセッサを割り当て，GHC に割り当てるプロセッサの台数を増やすことにより，図 9 に示す結果が得られた。図中の●印は，RBU が 1 台の場合の測定点であり，○印は，6 台の場合の測定点である。何れの場合も，プロットされた並列性能比は，GHC が 1 台の時の処理時間を 1 として，これに比べて，何倍速くなったかを示している。また，図中の実線は，理想的な並列性能比を表している。以上から，GHC が 9 台のとき，GHC が 1 台に比べて 9 倍近い並列性能が得られていることが分かる。

以上の結果から，推論により複数の制約式の組が作成されるような問題では，このような並列性能が期待できると言える。ここでは，等式に帰着される制約問題を扱ったが，不等式に帰着される制約問題では，掃き出し法を基本とする単体法のアルゴリズムで解けることから，不等式に帰着されるような問題でも，同じような並列性能が得られると考えている。

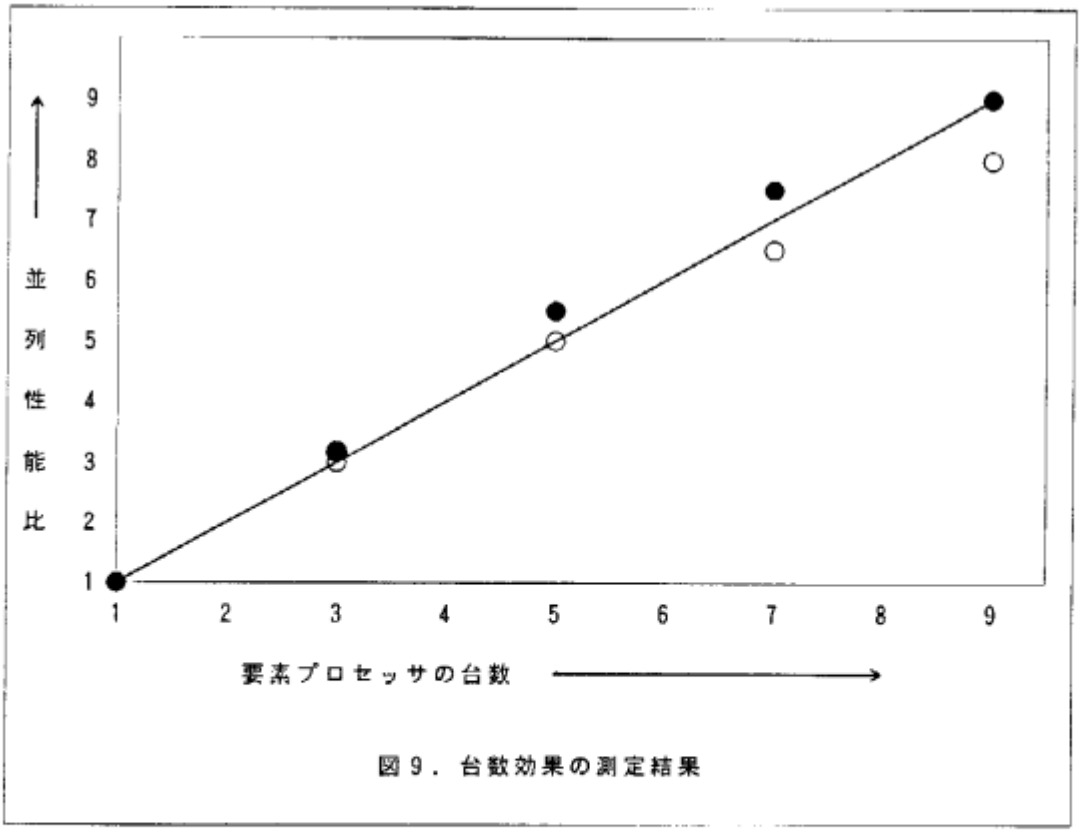


図 9．台数効果の測定結果

6．おわりに

本論文では，並列論理型言語GHC と逐次言語Ｃを使い，知識ベース指向の並列推論処理システムをメタプログラミングの考え方を基礎にして実現した。その中で，GHC でOR並列の知識ベース処理を行うために，知識ベースの変数とGHC の変数を区別した。即ち，知識ベースの変数を特殊な$ 変数で表し，$変数を扱う単一化処理や代入処理を実現した。実現に際しては，メモリー消費量を抑えるために，これらの$ 変数を扱う処理の部分をGHC で実現することを避け，逐次言語Ｃで実現した。また，本システムを拡張し，ｶｯﾄｼﾝﾎﾞﾙ が処理できるようにすることも容易であることも示した。さらに，大規模な知識ベース処理にも対処できるようにするために，ﾊｯｼｭやﾄﾗｲ で構造化された知識ベースに対して，GHC からアクセス可能にした。

最後に，本システムの評価を行うために，本システムを中核とする制約論理ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑ を作成し，測定を行った。測定においては，静電界の例題を採り上げた。測定の結果，良好な台数効果を得た。

今後の課題としては，部分計算〔Furukawa '88, Fujita '88〕やコンパイル等によるシステム全体の効率化や最適化等が考えられる。この研究は，今後，膨大な処理時間のかかる各種の応用システムを記号処理的な観点で，並列化し，高速な並列応用を実現するために有用な基本技術を提案している。また，本研究は，ﾘﾌﾚｸｼｮﾝ 〔Tanaka '90〕とも深い関わりがあり，並列処理にとって，本質的な要素を含んでいる。

【 謝辞 】

本研究に当たり，日頃から有益なコメントを下さった，ＩＣＯＴのＫＢＭ関係の方々及び㈱富士通研究所・林人工知能研究部長に深謝致します。

【 参考文献 】

[Fuchi '78] 淵一博: 問題解決と推論機構, 情報処理学会誌, Vol.11 No.10, 1978.

[Uchida '88] S.Uchida, K.Taki, K.Nakajima, A.Goto, and T.Chikayama: Research and Development of the Parallel Inference System in the Intermadiate Stage of the FGCS Project, Proc. of the International Conference on Fifth Generation Computer Systems 1988, 1988.

[Hattori '89] A.Hattori et al: A Hierarchical Structured Parallel Inference Machine, Parallel Computing '89, 1989.

〔Naito '90 〕 内藤 他: 高並列計算機の適用による大量遺伝子情報解析技術の実現, 並列処理ｼﾝﾎﾟｼﾞｳﾑ JSPP '90, 1990.

〔Fukumura '88〕 福村 他: 鋼材出荷計画ｴｷｽﾊﾟｰﾄｼｽﾃﾑと分枝限定法, ｵﾍﾟﾚｰｼｮﾝｽﾞ ﾘｻｰﾁ 1 月号, 1988.

[Bowen & Kowalski '81] K.A.Bowen and R.A.Kowalski: Amalgamating Language and Meta-Language in Logic Programming, TR 4/81, Syracuse University, 1981.

[Kitakami '84] H.Kitakami, S.Kunifuji, T.Miyachi, and K.Furukawa: A Methodology for Implementation of a Knowledge Acquisition System, Proc. of the 1984 International Symposium on Logic Programming, 1984.

〔Hayashi '73 〕 林 訳: ﾃﾞｼﾞﾀﾙ計算機の自動設計, 産業出版,1973.

〔Sato '90〕 H.Sato and M.Ikesaka: Particle Simulation on a Distributed Memory Highly Parallel Processor, Supercomputing in Nuclear Applications '90,1990.

[Kasif '83] S.Kasif, M.Kohli, and J.Minker: 'PRISM: A Parallel Inference System for Problem Solving', Proc. of the Logic Programming Workshop 83, 1983.

[Yang '87] R.Yang: P-Prolog: A Parallel Logic Programming Language, Series in Computer Science - Vol.9, World Scientific Publishing Co Pte LTD, 1987.

[Clark '84] K.L.Clark and S.Gregory: Notes on Systems Programming in Parlog, Proc. of the International Conference on Fifth Generation Computer Systems 1984, 1984.

[Shariro '84] E.Shapiro: System Programming in Concurrent Prolog, Proc. 11th Annual ACM Symp. on Principles of Programming Languages. ACM, 1984.

[Ueda '85] K.Ueda: Guarded Horn Clauses. Technical Report TR-103, ICOT, 1985.

[Ozawa '89] 小沢年弘, 細井聡, 服部彰: FGHCのメモリー使用特性と世代別ガーベジコレクション, 情報処理学会論文誌, Vol.30 No.9, 1989.

[Takeuchi '87] A.Takeuchi: 並列問題解決用言語ANDOR-Ⅱ, 日本ソフトウェア科学会第二回プログラムシンポジウム,1986.

[Kitakami '89] 北上始, 横田治夫, 服部彰: 知識処理向き並列推論エンジン, 電子情報通信学会研究会, CPSY, 88-50, 1988.

[Yokota '89] H.Yokota, H.Kitakami,and A.Hattori: Term Indexing for Retrieval by Unification. Proc. of 5th Int'l Conf. on Data Engineering,pp.313-320, 1989.

[Lassez '87] C.Lassez: Constraint Logic Programming, Fourth IEEE Symposium on Logic Programming, 1987.

[Heintz '87] N.Heintze, S.Michaylov, P.Stuckey: CLP(R) and Some Electrical Engineering Problems, Fourth IEEE Symposium on Logic Programming, 1987.

[Kitakami '89] 北上始, 横田治夫, 服部彰: 知識処理向き並列推論メカニズム, 情報処理学会第38回全国大会,1989.

[Kitakami '90] (22) 北上始, 横田治夫, 服部彰: 知識ベース指向の並列推論処理システム, 情報処理学会第41回全国大会,1990.

[Furukawa '88] K.Furukawa, A.Okumura, and M.Murakami: Unfolding Rules for GHC Programs, New Generation Computing, Ohmusha, LTD. 1988.

[Fujita '88] H.Fujita, A.Okumura, and K.Furukawa: Partial Evaluation of GHC Programs Based on the UR-set with Constraints. Proc. of the Fifth International Conference and Symposium on Logic Programming, 1988.

[Tanaka '90] J.Tanaka, Y.Ohta, and F.Matono: Overview of an Experimental Reflective Programming System: ExReps. FUJITSU Scientific and Technical Journal, Vol.26 No.1, 1990.