ICOT Technical Memorandom: TM-1054

TM-1054

データフロー解析による制約論理型言語の処理系の効率化検討

永井　保夫（東芝）、長谷川　隆三

May, 1991

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

データフロー解析による制約論理型言語の処理系の効率化検討

Dataflow Analysis of Constraint Logic Programs and Its Application to Source-Level Optimization for Program Improvement

永井 保夫
Yasuo NAGAI
(株) 東芝 情報処理・機器技術研究所
Toshiba Corp.

長谷川隆三
Ryuzou HASEGAWA
(財) 新世代コンピュータ技術開発機構
ICOT

This paper describes the dataflow analysis of the constraint logic programs and its application to the source-level optimization of programs. In order to improve the efficiency of the program execution, we introduce the dataflow analysis approach and execute the program optimization using information gathered through the analysis. First, we give the dataflow approach which uses the top-down analysis based on the SLD-refutation, without executing constraint solver. During this analysis, given program and goals (query), dataflow information is extracted from variable bindings and instantiation and propagated across each clause for the corresponding goals. On termination of the analysis, the substitution set and constraint set are collected, and instances of the constraint set are computed. Next, we consider these instances of constraint set as the structure of a matrix with a bipartite graph, called constraint graph. We explain the structural decomposition method of the constraint graph and its application to the source-level optimization for constraint logic program improvement using examples.

1 はじめに

制約論理型言語では制約という概念を論理型言語に導入することで，論理型言語と比較して，1) 対象となる構成要素やその属性間で成立する関係を関係表現や関数表現を用いてより宣言的に記述でき，2) プログラムの実行制御に関する表現もより宣言的に記述できるという特徴をもつ[7].

しかしながら，制約論理型言語の処理系は，制約の対象領域によっては必ずしも効率のよい処理を実現しているとはいえない場合がある．そこで，制約論理型言語の効率的な実行を実現するためには，制約論理型言語がもつ推論機構 (SLD- 導出) ならびに制約を取り扱う制約評価系の両者について考慮することが要求される．

本稿ではこのような問題点に対処するため，データフロー解析の利用による制約論理型言語における代数制約評価の効率化について述べる．

われわれは，問い合わせを入力として与えることにより得られる変数に関する束縛情報から制約集合をデータフロー解析により求め，求められた制約集合を構造解析し，制約間の依存関係情報を抽出する．この情報に基づいて制約の充足可能性に関する不必要な操作やバックトラックを減少させるため，制約評価の順序を決定する方法について検討する．

なお，これから説明する内容は，Buchberger アルゴリズムに基づいた代数制約評価系が提供されている制約論理型言語 CAL[6] を用いて具体的な検討をおこなっているものである．

2 準備

2.1 制約論理プログラムの操作的解釈

制約論理型言語の実行機構は論理型言語の特徴である操作的意味論に基づいた SLD- 導出 [9,10] に制約という概念を拡張したモデルであり，以下ではその定義について示す [6,7,8].

[定義] 確定節とは，$P \leftarrow P_1, P_2, \ldots, P_n; C_1, \ldots, C_m$ とする．ここで，$P_1, P_2, \ldots, P_n$ をリテラル部，$C_1, \ldots, C_m$ を制約部とする．
[定義] ゴール節とは，$\leftarrow P_1, P_2, \ldots, P_n;$ とする．ここで，$P_1, P_2, \ldots, P_n$ をリテラル部とする．
[定義] 導出節 (リゾルベント) R を $R =< RL; RC >$ とする．RL はリテラルに対する通常の論理プログラムの導出節であり，リテラル導出節とよぶ．RC は制約に対する導出節であり，導出により集められた制約集合の簡約形式である．
[定義] 代入 θ とは，$\{v_1/t_1, \cdots, v_n/t_n\}$ の形をした有限集合である．ここで，v_i は変数，t_i は v_i とは異なる項であり，変数 $v_1, \cdots, v_n$ は互いに異なる．各要素 v_i/t_i は，v_i の束縛とよばれる．
[定義] P を確定節，計算のある時点における導出節 R_n を $R_n =< L_1, L_2 \ldots L_m; RC >$ とする．
P 内の確定節 $P \leftarrow P_1, P_2 \ldots, P_n; C_1, C_2, \ldots, C_l$ とし，$P\theta$ と $L_1\theta$ が最汎単一化子 (mgu) θ をもつとき，θ を用いて，あらたな導出節 R_{n+1} は導出節 R_n と P から次の条件で導かれる．

- L_k は計算規則により選択された原子式である ($1 \leq k \leq m$).
- $RC' = (RC \cup C_1 \cup C_2 \cup \ldots C_l)\theta$ が可解 (solvable) ならば，R_{n+1} は P と R_n から
  $< (P_1, P_2 \ldots P_n, L_1, L_2 \ldots L_m)\theta; RC' >$ を導出する．

[定義] P を確定節，G をゴール節とする．導出が successful である導出列の最後の導出節 $R_n =< RL; RC >$ における リテラル部 RL が空 ($=\phi$) をとる．successful な導出における導出列の最後の導出節の制約部 RC を解制約という．

[定義] P を確定節とする．P の成功集合 S_P は次のように定義される．
$S_P = \{(\leftarrow P; C) \mid$ ゴール $\leftarrow P; \phi$ が解制約 C をもつ successful な SLD- 導出列を有する $\}$

2.2 制約論理プログラムの計算モデル

制約論理プログラムの実行は，制約が得られるたびに制約評価系が呼び出される．制約評価時に既に得られている制約と矛盾を生じるときにはバックトラックが起こり，あらたな制約が評価される．

このような制約論理型言語の実行機構は，次のような計算モデルにより実現される [7,8].

[制約論理プログラムの計算モデル]

入力: 制約論理プログラム P とゴール G
出力: P から G の successful な導出列が得られたとき解制約 $C\theta$ を出力; 失敗したときは，失敗を出力する

アルゴリズム:

```
1  リテラル部と制約部からなる導出節 R =< RL; RC > を入力ゴール G
   に初期化する; すなわち，R_n =< G; φ > とする;
2  while リテラル導出節 RL が空でない do
3      mgu θ により，L_1 と P が単一化できるようなリテラルゴール L_1
       をリテラル導出節から選択する;
4      確定節 P ← P_1, P_2, ..., P_n; C_1, C_2, ..., C_l を P から選択する;
5      リテラル導出節から L_1 を取り除き，P_1, P_2 ..., P_n を加える ;
6      if (RC ∪ C_1 ∪ C_2 ∪ ... C_l)θ が可解 (solvable)
         then これをあらたに制約導出節 RC とする;
7      θ を新しい導出節 R_n(=< RL; RC >) に適用する;
8  endwhile
9  if リテラル導出節が空
```

then 解制約 $C\theta$ を出力する
10 else 失敗を出力する.

CLP(R)[7] や CAL[6] に代表される制約論理型言語の大きな特徴として，制約評価系に対して与えられた制約が充足しているかどうかを調べるために，制約をインクリメンタルに評価し，冗長な計算を減らして，計算量をなるべく少なくする機能がある.

2.3 SLD 導出により生成される探索木

[定義] P をプログラム，G をゴールとする．計算規則による プログラム P に関するゴール G の探索木を次のように定義する.

- 探索木の根節点は G である.
- 探索木の各節点はゴールをあらわし，その中から 1 つのゴールが選択される.
- 導出節 $< L_1, L_2 \ldots L_m; C >$ を探索木のある節点とし，L_1 が計算規則により単一化可能な各入力節 $P \leftarrow P_1, P_2, \ldots, P_q; C_1, \ldots, C_r$ について子孫をもつ．子孫は導出節 $< (P_1, P_2 \ldots P_q, L_2 \ldots L_m)\theta; RC >$ と定義し，ここでは $RC = solve((C \cup C_1 \ldots \cup C_r)\theta)$，$\theta$ は L_1 と P の最汎単一化子とする.
- リテラル導出節が空節となる節点には子孫がない.

[定義] 探索木の葉は，リテラル導出節が空となるゴールに到達したときには，成功節点とよばれ，その節点にある選択されたゴールをそれ以上書き換えることができないときには，失敗節点とよぶ．成功節点は，探索木の根節点に対する解に相当する (ここでは制約導出形 RC が解制約に相当する).

3 制約論理型言語におけるデータフロー解析

制約論理型言語におけるデータフロー解析を用いた効率化処理では，論理型言語における関数性ならびに決定性の解析，述語のモード解析 [1,2,3,4] と同様な処理が考えられる [5].

関数性ならびに決定性の解析では，冗長な選択肢を生成しないで処理に必要となる時間ならびに空間領域が改善されることが期待される．また，述語のモード解析では制約評価に関する不必要な操作ならびにバックトラックを減少させるために，制約評価順序の決定が有効であると考えられる.

今回は非線形制約が取り扱える Buchberger アルゴリズムに基づいた代数制約評価系の提供された制約論理型言語 CAL に対して，前者の項目ではなく，後者の制約評価に関する不必要な操作の除去ならびにバックトラックの減少を目的としたデータフロー解析の利用について説明する.

3.1 トップダウン実行解釈に基づいたデータフロー解析

本節では，前節で述べた計算モデルに基づいた実行機構を前提とした制約論理プログラムのデータフロー解析について述べる．なお，実際の制約論理プログラムの実行解釈は，前節の計算モデルにおける任意のゴール選択を最左ゴールの選択とし，節の非決定的な選択を単一化可能な節の逐次選択とバックトラッキングに置き換えた実行モデル [10] に基づきおこなわれるとみなす.

データフロー解析はプログラムを直接実行しないで，実行時のふるまいに関する性質を予測したり，解析結果に基づいてコンパイラにおけるコードの最適化に用いられる [11].

われわれは，このようなデータフロー解析を用いて，トップレベルのゴールに対して，制約論理型言語の実行モデルから可能となる計算経路をトップダウンにトレースし，プログラムのふるまいに関する特徴を求める.

本処理では，入力であるプログラム P とゴール G $(P \cup \{G\})$ に対するプログラムの計算経路に対応する探索木 Tr をたどりながら，プログラム中でのデータ定義と参照の関係，すなわちデータフロー情報を解析し，出力として探索木上の成功節における制約集合 C と対応する代入 θ を求める．その場合，制約評価系が制約評価をおこなわない形で，制約論理プログラムを実行し，最終的に制約集合の代入例 $C\theta$ を求める．また，複数の成功節が存在する場合には，全解探索によりすべての制約集合，代入集合ならびに代入例を求める.

ここでは，プログラムにおけるゴール (述語) に関する情報ならびに節に関する情報の解析がおこなわれ，その概要は以下に示される.

[アルゴリズム]

入力: プログラム P ならびにゴール $G(P \cup \{Q\})$
出力: 探索木 Tr 上のすべての成功節における制約集合 C と代入 θ および代入例 $C\theta$ を求める
トップレベルの手続き:

```
Subst ← {};
Constr ← {};
analyze_goal(Goal, Subst, Constr);
apply_subst_to_constr(Subst, Constr, Instance);
```

```
procedure analyze_goal(Goal, Subst, Constr)
  begin
    for each clause Cl_i of Goal do;
    analyze_clause(Cl_i, Goal, Subst, Constr);
    endfor
  end;
```

ゴール (述語) 解析アルゴリズム

```
procedure analyze_clause(Cl, Goal, Subst, Constr)
  begin
    let Cl be of the form Head :- Body;
    if unify(Head, Goal, Subst1);
      then Subst ← Subst ∪ Subst1;
           analyze_body(Body, Subst, Constr);
      else failed_unification(Head, Goal, Subst) ;
  end;
```

節に関する解析

```
procedure analyze_body(Body, Subst, Constr)
  begin
    let Body be of the form Literal_Body; Constraint_Body ;
    if Literal_Body is empty
      then return Constr ← Constr ∪ Constraint_Body;
      else let Literal_Body be of the form p(X), LBody_Tail;
    generate_goal(p, Subst, Cp);
    analyze_goal(Cp, Subst, Constr1);
    analyze_body(LBody_Tail, Subst, Constr2);
    Constr ← Constr ∪ Constr1 ∪ Constr2;
  end;
```

ボディ部に関する解析

ゴールに関する解析 analyze_goal では，実行可能なすべての述語呼びだしについての変数束縛に関する情報を，各述語に対応するすべての節に伝播し，各述語の静的なフロー情報を求める．節に関する解析 analyze_clause では，節とゴールとのヘッドユニフィケーションをおこない，生成された代入情報をボディ部に伝播し，さらにサブゴールやヘッド間の変数共有情報を考慮してボディ部の解析 analyze_body をおこなう．なお，再帰計算の場合には，無限ループによる無駄な計算をしないように，ループの検出をおこなう.

[例題 1]

非線形代数制約を対象とした例題 1 について考える.

```
?-  f1(x2,x5), f2(x2,x5,x7), f3(x3,x6,x7), f4(x1,x7),
    f5(x3,x5,x8), f6(x1,x4,x7), f7(x6,x8), f8(x1,x4).
```

f1(X,Y) : $-2 * X^2 + Y = 2$.
f2(X,Y,Z) : $-X + Y^2 + Z = 1$.
f3(X,Y,Z) : $-X + Y + 2 * Z^3 = 4$.
f4(X,Y) : $-X + Y = 3$.
f5(X,Y,Z) : $-X^2 + 2 * Y + Z = 2$.
f6(X,Y,Z) : $-X^3 + Y + 3 * Z = 1$.
f7(X,Y) : $-X + Y^3 = 2$.
f8(X,Y) : $-X + Y = 4$.

図 1 に例題 1 のトップダウン実行解釈に基づくデータフロー解析から得られた証明木を示す.

3.2 制約集合の導出

制約論理プログラムの計算，すなわちゴールの証明は，ゴールに対応する探索木を実行モデルによって解釈し，証明木を求めることに相当する．その場合，制約集合をインクリメンタルに導出により集め，制約評価系 (上記の $solve(C \cup \cdots \cup RC)$) が制約の評価をおこなう．根節点から始まる証明木の各枝は，プログラム P によるゴール G の計算をあらわす．証明木の葉における リテラル導出節が空となるとき，成功節点とよび，その節点にある選択されたゴールがそれ以上書き換えられないときには失敗節点という．成功節点は，証明木の根節点における解に相当する．つまり，そこで

の解(制約)は，成功節点Kおける導出節 $\leftarrow \langle \phi; C\theta \rangle$ K対する $C\theta$ である．

本解析では，制約評価系Kよる制約評価 ($solve(C \cup \cdots \cup RC)$) をおこなわないで(解制約を求めるのではなく)，制約集合 $(C \cup \cdots \cup RC)$ を求める．例題1では，各ゴール f1(x2,x5)，f2(x2,x5,x7)，f3(x3,x6,x7)，f4(x1,x7)，f5(x3,x5,x8)，f6(x1,x4,x7)，f7(x6,x8)，f8(x1,x4) のリダクションKより，制約および代入が順次集められ制約集合と代入集合 $\{\theta_1, \theta_2, \ldots, \theta_8\}$ が得られる(図1)．最終的Kは，制約評価系Kよる制約の評価はおこなわれずK，制約集合 C の θ [1] $= \{$ 2*x2 + x5 =2，x2 + x5 +x7 =1，...，x1 + x4 =4 $\}$ Kよる代入例 $C\theta$ が求められる．本例題では，単一の制約集合のみを求めているが，問題Kよっては複数の制約集合を求めることが必要となる．

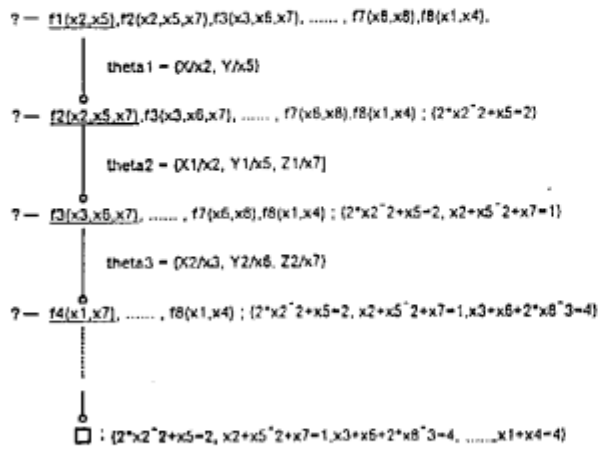

図1： 例題1のトップダウン解析

4 データフロー解析による制約論理型言語の処理系の効率化検討

制約論理型言語Kおける導出処理は第2節Kふしたようk リテラルK関する処理と制約K関する処理Kわけられる．

制約論理型言語の処理系を効率化するためKは，両者の導出処理をどのようK制御するかが重要な問題となる．たとえば，リテラルK関する導出処理ではリテラルゴールの選択規則ならびK選択されたリテラルと単一化する確定節の選択規則と確定節のボディ部のゴールの順序が全体の処理効率K影響を与える．一方，制約K関する処理では，リテラルの処理Kより集められた制約を制約評価系が解くので，その処理効率は制約の評価順序K依存したものとなる．

特K，後者の場合Kはわれわれの対象としている制約論理型言語CALの代数制約評価系がBuchbergerアルゴリズムK基づくため，アルゴリズムは次のような性質を有する．1) 非線形制約K対する基底計算で求められる多項式の次数が，最悪の場合で多項式環Kおける変数の数の二乗のべきとなるような複雑さとなり，2) 計算時間が制約の評価順序ならびK単項の優先順位K依存し，3) 解の係数成長が計算時間K非常K影響を及ぼす．このような性質から，制約評価系Kおける効率的な処理の実現が不可欠である[7,13]．

そこで，トップダウン実行K基づいたデータフロー解析Kより得られた制約集合を制約グラフとして表現し，グラフ論的手法を用いて制約評価系を効率化する手法[13]の適用Kついて検討する．本手法では，制約集合がもつ代数的構造を抽出し，その情報K基づいて求められた制約間の依存関係情報から，制約評価系K対する制御情報を生成し，効率的な処理を実現する．なお，現時点では解析の結果，複数の制約集合が存在する場合Kは，それぞれK対して効率化手法を適用する．

4.1 制約ならびに制約集合のグラフ表現

制約とは対象の構成要素およびその属性間で成立する関係を宣言的K記述したものである．制約は関係や関数Kよって表現される．

われわれは，このような関係や関数といった様々な形式をとる制約集合の代数構造をグラフKより抽象表現し，これを制約グラフとみなす[13,14]．制約集合を2部グラフ[13]を用いた制約グラフとして表現し，制約集合のもつ代数的構造情報を抽出する．

図2は，例題1の解析Kより求められた制約集合の代入例 $C\theta$ を2部グラフ表現したものである．

[1]代入 θ はそれぞれの代入の合成，すなわち $\theta_1 \circ \theta_2 \circ \cdots \circ \theta_8$ から求められる

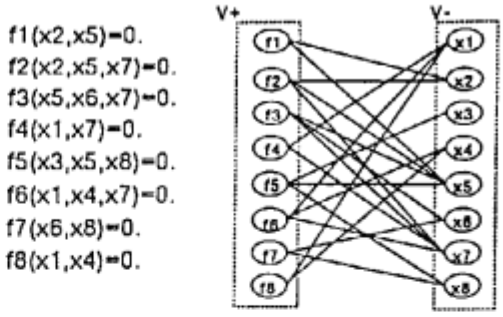

図2：例題1の解析により求められた制約集合の2部グラフ表現

4.2 抽象化された制約集合の構造分解ならびに整合性解析による代数制約評価の解釈

4.2.1 2部グラフのDM分解

2部グラフ $G = (V^+, V^-; E)$ KおけるDM分解とは既約成分への分解であり，半順序構造 $\prec$ をもつ $V(= V^+ \cup V^-)$ の部分集合の族である $\{G_-, G_+\} \cup \{G_i\}_{i=1}^n$ K完全正準分解することである．前者の $\{G_-, G_+\}$ を不整合部，後者の $\{G_i\}_{i=1}^n$ を整合部とよぶ．

制約集合を2部グラフ $G = (V^+, V^-; E)$ として表現し，2部グラフのDM分解をおこなうことKより，制約集合を表現するグラフが構造的K可解であるかどうかを判定し，可解であれば部分問題 $G_i = (W_i^+, W_i^-; E_i)$ $(i = 1, \ldots, n)$ K分割し，解を求めるための順序を決定できる．

4.2.2 制約集合の構造分解ならびに整合性解析による代数制約評価の解釈

制約グラフの整合性解析は次のような2部グラフのDM分解の定理から判定できる．

[定理]

2部グラフ $G = (V^+, V^-; E)$ のDM分解 $G_i = (W_i^+, W_i^-; E_i)$ $(i = \{1, \ldots, n\} \cup \{-, +\})$ K対して(a)から(c)が成り立つ．

(a) $W_+^- \neq \phi$ ならば，$|W_+^+| < |W_+^-|$ である．(G_+ の場合)

(b) $W_-^+ \neq \phi$ ならば，$|W_-^+| > |W_-^-|$ である．(G_- の場合)

(c) $|W_i^+| = |W_i^-|$ であって，G_i $(i = 1, 2, \ldots, n)$ は完全マッチングをもつ．

上記の定理から制約グラフの整合性を判定し，以下のような代数制約の評価K対する抽象的な実行解釈をおこなうことができる．具体的Kは，このような抽象化Kよる解釈は実数上で解を求める制約評価系では有効である．しかしながら，われわれの対象とするBuchbergerアルゴリズムk基づいた代数制約評価系では複素数上の解を求めるものであり，必ずしも抽象化Kよる解釈が有効とはいえない場合もあり考慮することが必要である．

(a)は構造的K可解でなく，制約(方程式)が不足した(under-constrained)状態であり，W_+^+ の属する方程式(制約)Kおいて未知数の数が方程式の数より多いため，解が一意K定まらない(不定である)ことを示している．

(b)も構造的K可解でなく，制約が矛盾または過合した(over-constrained)状態であり，未知数の数が方程式の数より少ないため，解が求まらない(不能である)ことを示している．

(c)は分解された各 G_i $(i = 1, \ldots, n)$ が完全マッチングをもち，G も完全マッチングをもつため，構造的K可解であることを示している．

一方，構造解析Kより求められる制約間の依存関係情報は，後述する制約評価の処理効率の向上K有効である．$\mathcal{G} = \{G_i\}$ を半順序 $\prec$ K矛盾しないようK並べ換え，$W_-^- = W_+^+ = \phi$ かつ $\mathcal{G}$ Kよって得られる有向グラフをブロック三角行列の形(スパース構造)K表現できれば，この情報を求めることができる．その結果，$(W_n^+, W_n^-), (W_{n-1}^+, W_{n-1}^-), \ldots, (W_2^+, W_2^-)$ (W_1^+, W_1^-) Kが対応するブロックごとK分割して効率的K解くことが期待できる．

図3は図2で表現された2部グラフをDM分解した結果，G_1, G_2, G_3 という3つの部分グラフK構造分解され(これは(c)K該当する)，構造的K可解であることを示す．この結果から，各部分問題K関する制約の集合ごとK求解(制約評価)していけば，効率的な処理が期待される．

4.3 抽象化された制約集合の構造情報を用いた代数制約評価の効率化

4.3.1 制約の評価順序ならびに変数の優先順位の決定

制約の評価順序指定および変数の優先順位指定Kより，代数制約評価系の効率的な処理手法Kついて述べる．本手法では，制約評価系K対する制

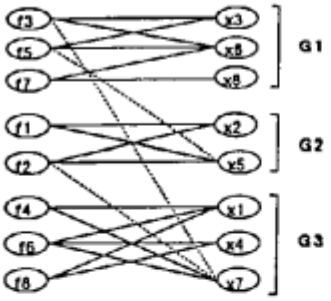

図 3: 例題1のDM分解結果

循情報を与えることにより，制約評価において無駄な計算を避け，バックトラックの減少させ，処理の効率化を目的としている．この方法は，われわれの対象としているBuchbergerアルゴリズムに基づいた代数制約評価に対しては，解制約であるグレブナ基底計算において冗長な計算 (S-多項式) を減らし，解の係数成長をなるべく抑制するという点において有効である [13,14].

DM分解によってブロック三角化をおこない，ブロック上三角化行列を求める．まず，求められた依存関係情報に基づき，部分問題 G_3 からはじめて，G_2, G_1 という順序に従って，部分問題ごとに制約評価ができるようにゴール列を並べ変える (図4).

次に，変数間の優先順位を部分問題 G_n の変数集合 $\ll$ 部分問題 G_{n-1} の変数集合 $\ll \cdots$ 部分問題 G_1 の変数集合となるようにpre述語を指定し (図5)，代数制約評価系の基本処理である項書き換え操作を制御し，グレブナ基底を効率的に求める．

4.3.2 効率化手法の適用

ここでは，非線形制約を取り扱ったいくつかの問題に対する効率化手法の適用実験とその評価について述べる (表1)．実験は逐次推論マシンPSI-II上で，制約論理型言語CALバージョン1.4を用いておこなった．なお，データフロー解析ならびに制約グラフの構造解析に関する処理はPSI-II上でESP言語を用いて実現中であり，現時点では人手による解析[2]をおこなった．

例題1において，制約の評価順序ならびに変数の優先順位を指定したプログラムを図4ならびに図5に示す．表1に示されるように効率化手法が有効であることが示されている．

```
:- f4(x1,x7),
   f6(x1,x4,x7),
   f8(x1,x4).        } G3

   f1(x2,x5),
   f2(x2,x5,x7),     } G2

   f3(x3,x6,x7),
   f5(X3,X5,X8),
   f7(x6,x8).        } G1
```

図 4: 制約の評価順序の指定 (ゴールの並び換え)

```
pre(x3,100),
pre(x6,100),
pre(x8,100),
pre(x2,99),
pre(x5,99),
pre(x1,98),
pre(x4,98),
pre(x7,98).
```

図 5: 述語 pre による変数の優先順序の指定

例題2は熱交換器の設計問題 [12] を対象とし，例題1とは異なった再帰データ構造を用いたプログラムとして表現されている [13]．例題3および例題4は，幾何定理証明問題を対象とし，前者が3垂線定理を，後者が9点円定理を示す [14].

[2]制約グラフの構造解析はSUN3上のSicstus Prologにより実現されたシステムを利用した．

対象とした問題	適用前	適用後
例題1 (8制約, 8変数)	610	68
例題2 (熱交換器設計 :12制約, 10変数)	281279	1322
例題3 (3垂線定理 :5仮説, 8変数)	7565	2371
例題4 (9点円定理 :9仮説, 12変数)	3524098	12753

表1: 非線形制約を対象とした問題に対する適用結果 (単位:msec)

これらの問題は，制約集合のグラフ表現がスパース (疎) な構造をとるものであり，効率化手法が有効な場合である．それに対して，グラフ構造がスパースでない (密である) 場合には，本手法はあまり有効ではなかった．

4.4 今後の課題について

本研究では，以下のような項目について，今後の課題として検討していく必要がある．

- 抽象解釈の枠組の適用
- 他領域 (例えば，ブール領域) の制約評価に対する制約の構造解析手法の有効性
- 複数の制約集合が求められた場合の有効な取り扱い
- インクリメンタルな制約集合の構造解析アルゴリズム
- 別名 (aliasing) の取り扱い

5 まとめ

ゴールを入力として与えた場合，節および述語に対する入出力情報と述語，関数，制約などからデータ依存関係を解析し，全体のプログラムの実行順序を推論し，より実行効率のよいソースレベルコードを生成することを目的として，データフロー解析による制約論理型言語処理系の効率化について検討をおこなった．その結果，制約集合のグラフ表現がスパース (疎) な構造をとる問題に対しては，効率化手法が特に有効であることが明確となった．今後は，本手法を導入したCALのソースレベルのプログラム最適化ツールを実現し，プログラムの解析時間ならびに実行時間に関する評価をおこなう予定である．

謝辞

本研究は第5世代コンピュータプロジェクトの一環として行なわれた．本研究の機会を与えてくださり，常にご指導いただいたICOTの渕一博所長，古川康一研究次長，新田克己第7研究室室長ならびに，有益なコメントをいただいた相場亮第4研究室長代理に深く感謝いたします．また，CALを利用するにあたり，いろいろ教えて頂いた第4研究室の皆様に感謝いたします．

参考文献

[1] S.K. Debray, Static Inference of Modes and Data Dependencies in Logic Programs, *ACM Trans. on Programming Languages and Systems*, Vol.11, No.3, July 1989
[2] M. Bruynooghe, et al. Abstract Interpretation: Towards the Global Optimization of Prolog Programs, *Proc. of SLP '87*, 1987
[3] H. Mannila, E. Ukkonen, Flow Analysis of Prolog Programs, *Proc. of SLP '87*, 1987
[4] T. Kanamori, et al., Logic Program Analysis by Abstract Hybrid Interpretation, ICOT Technical Report TR-485, 1989
[5] K. Marriott, H. Sondergaard, Analysis of Constraint Logic Programs, *Proc. of NACLP '90*, 1990
[6] K. Sakai and A. Aiba, CAL: A Theoretical Background of Constraint Logic Programming and its Application, *J. Symbolic Computation 8*, 1989
[7] J. Cohen, Constraint Logic Programming Languages, *Comm. of the ACM*, Vol.33, No.7, 1990
[8] M. Dincbas, Constraint, Logic Programming and Deductive Database, *Proc. of France-Japan Artificial Intelligence and Computer Science Symposium 86*, 1986
[9] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1984 (邦訳: 論理プログラミングの基礎, 佐藤雅彦他訳)
[10] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986 (邦訳:Prologの技芸, 松田利夫訳)
[11] M.S. Hecht, *Flow Analysis of Computer Programs*, North-Holland, 1977
[12] E.J. Henry and R.A. Williams, *Graph Theory in Modern Engineering, Computer Aided Design, Control, Optimization, Reliability Analysis*, Academic Press, 1973.
[13] 永井保夫, 制約グラフの構造分解および整合性解析による代数制約評価系の効率化についての検討, 人工知能学会研資 SIG-F/H/K-9001-12, 1990
[14] 永井保夫, 代数制約の構造情報を用いた幾何定理証明の効率化手法の検討, 情報処理学会第42回全国大会 6F-1, 1991