ICOT Technical Memorandom: TM-1106

TM-1106

制約論理型言語における プログラム解析システムの実現検討

永井 保夫 (東芝)、長谷川 隆三

September, 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

制約論理型言語におけるプログラム解析システムの実現検討

Towards Program Analysis System for Constraint Logic Program and Its Implementation

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

本稿では，まず制約論理型言語を対象としたプログラム解析システムの概要ならびに主要な処理であるデータフロー解析，制約集合の構造解析，制約の評価順序および変数の優先順序に基づいたソースコードの最適化について述べる．次に，最大マッチング計算に有効なアルゴリズムの導入による効率的な DM 分解 (制約集合の構造解析) 処理への拡張ならびにそれに適した解析システムの実現方式について検討する．ここでは，制約論理型言語 CAL の代数制約に対する解析処理について例題を用いて説明する．最後に，いくつかの例題に対する解析の適用結果ならびに本解析アプローチの有効性について示す．

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 [5].

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．このような場合には，制約論理型言語の処理系においてリテラルに関する導出処理と制約に関する導出処理 [6] をどのように制御するかが重要な問題となる．たとえば，リテラルに関する導出処理ではリテラルゴールの選択規則ならびに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に影響を与える．一方，制約に関する処理では，リテラルの処理に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に依存する．

特に，われわれの対象としている制約論理型言語 CAL の代数制約評価系は Buchberger アルゴリズムに基づくため，制約の評価順序を考慮した効率的な処理の実現が不可欠であると考えられる [4,5,12].

そこで，制約論理型言語の処理系の効率的処理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 [7,8]) 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．論理型言語のプログラム解析に関する研究では，関数性ならびに決定性の解析や述語のモード解析 [1,2] などの手法が盛んに検討されており，制約論理型言語における適用検討もなされている [3].

われわれはこのような手法を考慮し，制約論理型言語を対象としたプログラム解析システムを開発中である．本システムでは，Buchberger アルゴリズムに基づいた代数制約評価系が提供された制約論理型言語 CAL[4] により記述されたプログラムを解析し，制約評価に関する不必要な操作を除去することにより処理の効率化をおこなう．その解析処理では，制約論理型言語の計算モデル (SLD- 導出)[5,6] に基づいて，与えられた問い合わせ (ゴール) から，プログラムのふるまいを予測し，制約集合を求める．得られた制約集合を構造解析し，制約間の依存関係を抽出し，処理の効率化を目的とした制約評価の順序を決定する．

ところで，これらの解析処理のなかで，制約集合の解析，特に最大マッチングの計算が全体の処理に対して比重を占めていると考えられ，システム実現を考えた場合，効率的なアルゴリズムが要求される．われわれは Hopcroft と Karp により与えられた効率的な計算アルゴリズムを用いた解析システムの実現方式について検討している．

本稿では，まずプログラム解析システムの概要ならびに主要な処理であるデータフロー解析，制約集合の構造解析，制約の評価順序および変数の優先順序に基づいたソースコードの最適化について述べる．次にインクリメンタルな最大マッチングの計算アルゴリズムの導入による効率的な DM 分解処理への拡張 (制約集合の効率的な構造解析) ならびにそれに適した解析システムの実現方式について説明する．ここでは特に，制約論理型言語 CAL の代数制約に対する解析処理について例題を用いて示す．最後に，いくつかの例題に対する解析の適用結果について示す．

2 プログラム解析システムの処理概要

われわれは，トップダウン実行に基づいたデータフロー解析に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を提案した [14]．今回提案するプログラム解析システムでは，この手法の導入により，制約集合がもつ代数的構造を抽出し，その情報に基づいて求められた制約間の依存関係情報から，制約評価系に対する制御情報を生成し，ソースコードレベルの最適化をおこなう．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には，それぞれに対して最適化をおこなう．

プログラム解析システムの処理概要は，図 1 に示されるようなデータフロー解析による制約集合の導出，導出された制約集合の構造解析，制約の評価順序および変数の優先順位に基づいたソースコードの最適化という 3 つの処理からなる．以下ではそれぞれの処理の内容について説明する．

2.1 データフロー解析による制約集合の導出

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 [9].

データフロー解析を用いた制約論理プログラムの制約集合の導出処理では，トップレベルのゴールに対して，制約論理型言語の SLD 導出に基づいた計算モデル (実行機構) [5,6] 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，特にここでは制約集合を求める．

具体的には，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に関する定義と参照の関係，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路における制約集合 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，複数の成功路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．この制約集合の導出では，制約評価系による制約評価 ($solve(C \cup \cdots \cup RC)$) 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 [14].

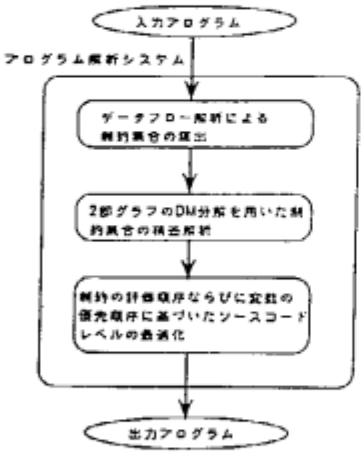

図1：制約論理型言語を対象としたプログラム解析システムの処理フロー

[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.

```
?-   example1.

example1. :- 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.
```

制約集合の導出処理により，例題 1 では example1 の各サブゴール 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 [8] がおこなわれ，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．最終的には，制約評価系による制約の評価はおこなわれずに，制約集合 C の $\theta = \theta_1 \circ \theta_2 \circ \cdots \circ \theta_8 =$ { 2*x2 + x5 =2, x2 + x5 +x7 =1, ..., x1 + x4 =4 } による代入例 $C\theta$ 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，問題によっては複数の制約集合を求めることが必要となる.

2.2 2部グラフのDM分解を用いた制約集合の構造解析

2.2.1 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に記述したものである．制約は関係や関数によって表現される.

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により抽象表現し，これを制約グラフとみなす [12,13,14]．制約集合を 2 部グラフ [12,14] を用いた制約グラフとして表現し，DM 分解により制約集合のもつ代数的構造情報を抽出する.

2.2.2 2部グラフのDM分解

2 部グラフ $G = (V^+, V^-; E)$ における 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 $\{G_-, G_+\} \cup \{G_i\}_{i=1}^n$ に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により，制約集合を表現するグラフが構造的に可解であるかどうかを判定し，可解であれば部分問題 $G_i = (W_i^+, W_i^-; E_i)$ $(i = 1, \ldots, n)$ に分割し，解を求めるための順序を決定できる.

(a) DM 分解の処理概要

次に 2 部グラフの DM 分解の処理概要について説明する．具体的なアルゴリズムは，図 2 に示される．(1) では，2 部グラフ $G = (V^+, V^-; E)$ の最大マッチング M を求める．(2) では，最大マッチング M の各エッジの逆向きエッジを G に追加して得られる補助グラフ $G_M = (V^+, V^-; \tilde{E})$ $(\tilde{E} = E \cup \{(u,v) \mid (v,u) \in M\})$ を求める．(3) では，不整合部 $\{G_-, G_+\}$ に対する処理をおこなう．まず，$V^+ - \partial^+ M$ の点を始点とする補助グラフ G_M の有向道の終点全体を $U_{(-)}$ とし，$V^- - \partial^- M$ の点を終点とする G_M の有向道の始点全体を $U_{(+)}$ としたとき，$U_{(+)}, U_{(-)}$ により G の部分グラフ G_-, G_+ を誘導する．(4) では，$G_M - (U_{(-)} \cup U_{(+)})$ を強連結成分 $G_i = (W_i^+, W_i^-; E_i)$, $(i = 1, 2, \ldots, n)$ に分解する．$G' = \{G_i\}_{i=1}^n$ とする．(5) では，得られた $\{G_-, G_+\} \cup G'$ を半順序関係と矛盾しないように並べ換える.

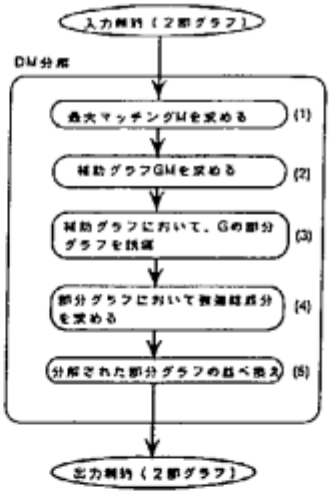

図2：DM分解の処理フロー

(b) DM 分解の効率的な計算アルゴリズムへの拡張

図 2 の DM 分解アルゴリズムでは，(1) の 2 部グラフの最大マッチング M の計算の手間が全体の処理に対して支配的である．われわれは，このような DM 分解の計算の手間を減少させるために，従来提案されてきた $O(mn)$ のアルゴリズムのかわりに，非常に効率的な Hopcroft & Karp アルゴリズム (HK アルゴリズム) を用いた DM 分解処理への拡張について検討する.

HK アルゴリズムは，マッチングと増加路という概念の導入により，2 部グラフの最大マッチングを $O(m\sqrt{n})$ 時間で求めるアルゴリズムである ($G = (V; E)$ において，$|V| = n$, $|E| = m$ とする)[10]．これは，いままでに求められているマッチングに関して増加路を探索し，このマッチングに基づいて解である最大マッチングをインクリメンタルに求めるアルゴリズムであり，その概要は以下に示される.

- [ステップ 0] $M \leftarrow \phi$ とする.

- [ステップ1] マッチング M に関する増加路が存在しなければ，現時点での M が最大マッチングである．もし存在すれば，長さが最小でかつ互いに点を共有しない増加路の最大集合 $\{Q_1, Q_2, \cdots, Q_l\}$ を求める．
- [ステップ2] $M \leftarrow M \oplus Q_1 \oplus Q_2 \oplus \cdots \oplus Q_l$ として，ステップ1へ戻る．なお，集合 M と P に対する演算 $M \oplus P \stackrel{\text{def}}{=} (M - P) \cup (P - M)$ とする．

われわれの提案するDM分解の拡張アルゴリズムは，図2の処理において，(1)の部分に対してHKアルゴリズムを適用し，インクリメンタルに最大マッチングを算出し，得られた最大マッチングを用いて(2)から(5)までの処理を順次実行し，最終的に分解された制約集合を求めるものである．

このような拡張アルゴリズムにより後述する制約集合の構造解析を実現するには，制約集合を(データフロー解析により)すべて導出してから，それらの構造解析をおこなう方法と制約が導出されるたびにインクリメンタルに最大マッチングを求めることにより構造解析をおこなう方法の2つが考えられる．ここでは制約論理型言語の枠組に容易に取り入れることができる後者のインクリメンタルに最大マッチングを求める方法を選択する．

2.2.3 制約集合の構造解析

次のような2部グラフのDM分解の定理から，制約集合の構造解析がおこなわれる．

[定理]

2部グラフ $G = (V^+, V^-; E)$ のDM分解 $G_i = (W_i^+, W_i^-; E_i)$ $(i = \{1, \ldots, n\} \cup \{-, +\})$ に対して(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に対する抽象的な実行解釈をおこなうことができる．具体的には，このような抽象化に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アルゴリズムに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，必ずしも抽象化に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)は構造的に可解でなく，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)において未知数の数が方程式の数より多いため，解が一意に定まらない(不定である)ことを示している．

(b)も構造的に可解でなく，制約が矛盾または競合した(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，構造的に可解であることを示している．なお，構造的に可解であるとは，2部グラフ G 上に完全マッチングが存在することであると定義する．

制約集合の構造解析処理では，$\mathcal{G} = \{G_i\}$ を半順序 $\prec$ に矛盾しないように並べ換え，$W_-^- = W_+^+ = \phi$ かつ $\mathcal{G}$ に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)に表現することにより，制約間の依存関係情報を求める．その結果，$(W_n^+, W_n^-), (W_{n-1}^+, W_{n-1}^-), \ldots, (W_2^+, W_2^-), (W_1^+, W_1^-)$ に対応するブロックごとに分割して効率的に制約評価をおこなうことが期待できる．このような制約間の依存関係情報は，次節の制約評価の処理効率の向上のためのソースコードの最適化に利用される．

なお，図3は例題1に対してデータフロー解析をおこない，導出された制約集合を構造解析した結果を示す．

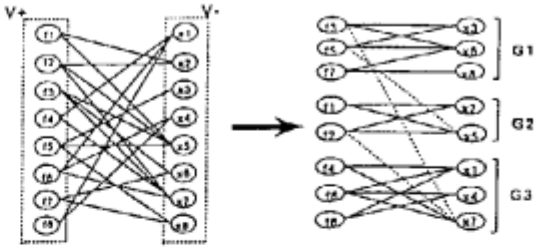

図3: 例題1の制約集合の2部グラフ表現とDM分解の結果

2.3 制約の評価順序および変数の優先順位に基づいたソースコードの最適化

ソースコードの最適化処理では，制約集合の構造解析により求められた制約間の依存関係情報から制約の評価順序ならびに変数の優先順位を求め，これに基づいて制約評価系に対する制御情報を生成し，ソースコードの変換をおこなう．その結果，最適化されたソースコードの実行時に，制約評価において無駄な計算を避け，処理の効率化が期待される．この最適化処理では，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，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[12,13,14]．

以下では例題1による具体的な処理について説明する．

まず，DM分解によって求められたブロック上三角化行列から得られる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，変数間の優先順位が部分問題 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．

結果として，求められた制約の評価順序および変数の優先順位に基づいて生成されたプログラムを図4ならびに図5に示す．

```
example1 :-
    f4(x1, x7),
    f6(x1, x4, x7),      } G3
    f8(x1, x4),
    f1(x2, x5),          } G2
    f2(x2, x5, x7),
    f3(x3, x6, x7),
    f5(X3, X5, X8),      } G1
    f7(x6, x8).
```

図4: ゴールの並び換えによる制約の評価順序の決定

```
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

3 プログラム解析システムの実現検討

本解析システムの実現方式には，インタプリタ(メタレベルインタプリタ)による方式とPROLOGインタプリタによ

る方式が考えられるが，われわれはシステム実現の容易性を考慮して，後者のPROLOGインタプリタ方式を採用した．

本方式では，トランスレータが解析対象となるプログラム(例題1)を入力とし，図6に示されるプログラムを出力する．このようにトランスレータにより生成されたプログラムの実行により，プログラム解析がおこなわれる．図6のプログラムでは，HKアルゴリズムにより最大マッチングを求める処理が制約ソルバーとみなせ，制約論理型言語のアーキテクチャが実現されていると考えることもできる．

次に，図6の解析プログラムにおける主要述語の処理について説明する．述語analysis_example1/2は解析プログラムのトップレベルのゴールをあらわす．そのサブゴールである述語example1_goal/2では，第一引数が入力である初期マッチングϕを，第二引数が求められる最大マッチングMを示しており，図2のDM分解の処理における(1)の部分に対応する処理をおこなう．述語dm_analysis_aux/2では，求められたMを入力として，(2)から(5)までの一連の処理をおこない，最終的に構造分解された制約集合を出力する．述語optimize/3では，構造分解された制約集合から求められた制約の評価順序および変数の優先順位に基づいてソースコードの最適化をおこなう(その際，制約の評価順序と変数の優先順位に関する情報をユーザに表示する)．その第一引数は入力ファイル(解析されるプログラム)名，第二引数は出力ファイル(最適化されるプログラム)名をあらわす．述語find_matching/3では，順次入力された制約からインクリメンタルに2部グラフの最大マッチングを求める．第一引数は入力制約を，第二引数はいままでに求められた最大マッチング，第三引数は入力制約により新しく求められる最大マッチングを示す．

```
analysis_example1(Ifile,Ofile) :-
     example1_goal([],B),
     dm_analysis_aux(B,G),
     optimize(Ifile,Ofile,G).
example1_goal(A,B):-
     f1(x2,x5,A,D),
     f2(x2,x5,x7,D,E),
     f3(x3,x6,x7,E,F),
     f4(x1,x7,F,G),
     f5(x3,x5,x8,G,H),
     f6(x1,x4,x7,H,I),
     f7(x6,x8,I,J),
     f8(x1,x4,J,B).
f1(A,B,C,D):- find_matching(''(2*A^2+B=2),C,D).
f2(A,B,C,D,E):- find_maching(''(A+B^2+C=1),D,E).
f3(A,B,C,D,E):-find_matching(''(A+B+2*C^3=4),D,E).
f4(A,B,C,D):- find_matching(''(A+B=3),C,D).
f5(A,B,C,D,E):- find_matching(''(A^2+2*B+C=2),D,E).
f6(A,B,C,D,E):- find_matching(''(A^3+B+3*C=1),D,E).
f7(A,B,C,D):- find_matching(''(A+B^3=2),C,D).
f8(A,B,C,D):- find_matching(''(A+B=4),C,D).
```

図6: トランスレータにより生成された例題1の解析プログラム

4 アプリケーション

いくつかの非線形制約を取り扱った問題を記述したプログラムに対して，本プログラム解析システムを用いてプログラムの実行時間に関する評価をおこなった(表1)．ここでは逐次推論マシンPSI-II上で，制約論理型言語CALバージョン1.4を用いて評価をおこなった．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，今回のプログラム解析では，制約の導出ならびに制約集合の構造解析処理をSUN3上の解析システムを用い，最適化処理を人手によりおこなった．

例題2は熱交換器の設計問題[11]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．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[13]．

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．

5 まとめ

われわれは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，特に制約に関する依存関係を解析し，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たプログラム解析システムを提案し，必要となる処理の概要について述べ，さらに，システムの実現方式について検討した．現在，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逐次推論マシンPSI II上でESP言語を用いて実現中であり，プログラムの解析時間に関する評価を早急に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 and D.S. Warren, Automatic Mode Inference For Logic Programs, *The Journal of Logic Programming 1988:5*, 1988

[2] H. Mannita, E. Ukkonen, Flow Analysis of Prolog Programs, *Proc. of SLP '87*, 1987

[3] K. Marriott, H. Sondergaard, Analysis of Constraint Logic Programs, *Proc. of NACLP '90*, 1990

[4] K. Sakai and A. Aiba, CAL: A Theoretical Background of Constraint Logic Programming and its Application, *J. Symbolic Computation 8*, 1989

[5] J. Cohen, Constraint Logic Programming Languages, *Comm. ACM*, Vol.33, No.7, 1990

[6] M. Dincbas, Constraint, Logic Programming and Deductive Database, *Proc. of France-Japan Artificial Intelligence and Computer Science Symposium 86*, 1986

[7] 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)

[8] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986 (邦訳:Prologの技芸, 松田利夫訳)

[9] M.S. Hecht, *Flow Analysis of Computer Programs*, North-Holland, 1977

[10] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM J. Comput., Vol.2, No.4*, 1973

[11] E.J. Henry and R.A. Williams, *Graph Theory in Modern Engineering, Computer Aided Design, Control, Optimization, Reliability Analysis*, Academic Press, 1973.

[12] 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

[13] 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

[14] 永井保夫, 長谷川隆三, 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, 1991年度人工知能学会全国大会 12-5, 1991