制約論理型言語における制約集合の構造解析による代数制約ソルバーの効率化についての検討

永井保夫*

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［6］。

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［7］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ブフバーガ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［5］［6］［14］。

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［8］［9］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］［3］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［11］。

本論文ではこのような問題点に対処するために、プログラム解析に関する研究でおこなわれている解析手法ならびにグラフ論的手法の適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検討をおこなった。

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に、制約評価の順序を決定する方法について検討した。

さらに、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部分試作した。本システムでは、ブフバーガ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［5］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

*東京情報大学総合情報学部情報システム学科助教授
2002年11月12日受理

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も検討をおこなった。

以下では、まず制約論理型言語におけ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代数制約ソルバーの効率化手法と制約論理型言語CALを用いて定式化した問題への適用実験について述べる。次に、これらの効率化手法に基づいたソースコードの最適化について述べる。さらに、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。最後に、制約論理型言語CALの代数制約により記述された例題に対する解析システムの適用結果と考察について示す。

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$R=\langle RL;RC\rangle$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

$R_n=\langle L_2.L_2,\ldots L_m;RC\rangle$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）$\theta$をもつとき、$\theta$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（$1\leq k\leq m$）。
- $RC'=(RC\cup C_1\cup C_2\cup\ldots C_1)\theta$が可解（solvable）ならば、$R_{n+1}$は$P$と$R_n$から

$\langle(P_1.P_2\ldots P_n.L_1.L_2\ldots L_m)\theta\ ;RC'\rangle$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$R_n=\langle RL;RC\rangle$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。

$S_p=\{(\leftarrow P;C)\mid \leftarrow P;\phi$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［6］［7］。

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**（R_C∪C_2∪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。

図1：CLPの計算モデル

抽象インタプリタにより行われる計算は、図1のようなゴールのリダクションによって進行していく。ゴールのリダクションでは、ある計算段階において、導出節中のリテラル部のゴールが選択され、そのゴールと単一化するようなヘッドをもつプログラム節が選ばれる。選ばれた導出節中のリテラル部のゴールは選ばれた節の本体のリテラル部で置き換えられ、制約部の制約集合は選ばれた節の本体の制約集合が加えられ全体の制約集合の可解性が判定される。このようにして節のヘッドと最汎単一化代入を適用して得られる新しい導出節を用いて計算が進められる。導出節のリテラル部が空のとき計算は停止する。

CLP（$\mathcal{R}$）［6］やCAL［5］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〈$(P_1,P_2,\ldots P_qL_2,\ldots L_m)\theta;RC$〉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こ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］と同様な処理が考えられる［11］。
関数性ならびに決定性の解析では、冗長な選択枝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ブフバーガ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［9］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［10］。

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T_r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、その概要は図2に示される。

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]

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).
```

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

[例題2]

非線形代数制約を対象とした例題2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).
```

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

[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T_r上のすべての成功路における制約集合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
```

図2：トップダウン実行解釈に基づいたデータフロー解析

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*($C \cup \ldots \cup RC$)）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、成功節点における導出節←〈ϕ ： $C\theta$〉に対する$C\theta$である。

本解析では、制約評価系による制約評価（*solve*($C \cup \ldots \cup RC$)）をおこなわないで（解制約を求めるのではなく）、制約集合（$C \cup \ldots \cup RC$）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)**のリダクションにより、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θ[1]$=\{$**2*x2 + x5 = 2, x2 + x5 +x7 = 1, ..., x1 + x4 = 4**$\}$ による代入例Cθ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。

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

制約論理型言語における導出処理は前述したようにリテラルに関する処理と制約に関する処理にわけられる。

制約論理型言語の処理系を効率化するためには、両者の導出処理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

[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

価系が解くので、その処理効率は制約の評価順序に依存したものとなる。

特に、後者の場合に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ブフバーガアルゴリズムに基づくため、アルゴリズムは次のような性質を有する。1) 非線形制約に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に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に単項の優先順位に依存し、3) 解の係数成長が計算時間に非常に影響を及ぼす。このような性質から、制約評価系における効率的な処理の実現が不可欠である［6］［14］。

そこで、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［16］の適用について検討する。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、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には、それぞれに対して効率化手法を適用する。

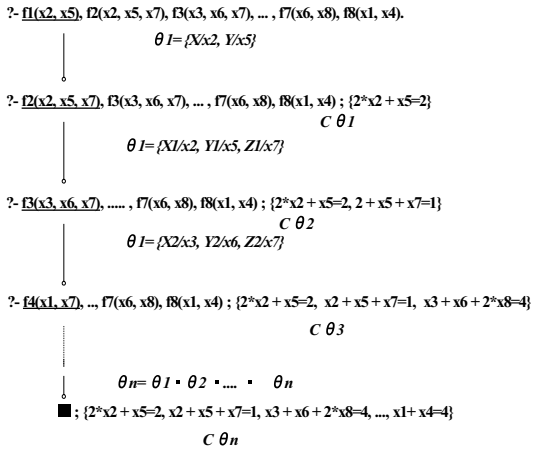

図3：例題1のトップダウン解析

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［14］［16］。制約集合を2部グラフ［14］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4は、例題1の解析に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。

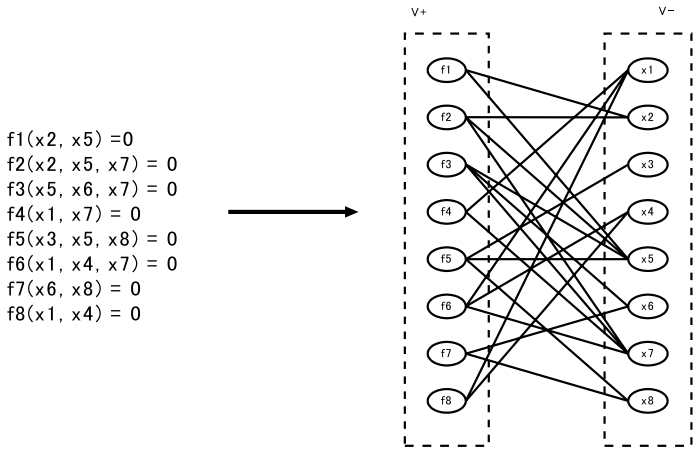

図4：例題1の2部グラフ表現

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$G=(V^+,V^-;E)$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,…,n）に分割し、解を求めるための順序を決定できる。

1. 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5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,…,$n$）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。

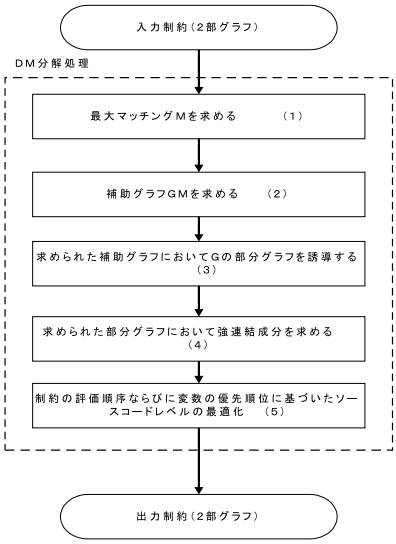

図5：DM分解のアルゴリズムの概要

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$G_i=(W_i^{+},W_i^{-};E_i)$（$i=\{1,\ldots,n\}\cup\{-,+\}$）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ブフバーガ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。

一方、構造解析に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に有効である。$\mathcal{G}=\{G_i\}$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^-)，$(W_{n+1}^+,W_{n+1}^-),\ldots,(W_2^+,W_2^-)$，$(W_1^+,W_1^-)$ に対応するブロックごとに分割して効率的に解くことが期待できる。

図6は図4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つの部分グラフに構造分解され（これは（c）に該当する）、構造的に可解であることを示す。この結果から、各部分問題に関する制約の集合ごとに求解（制約評価）していけば、効率的な処理が期待される。

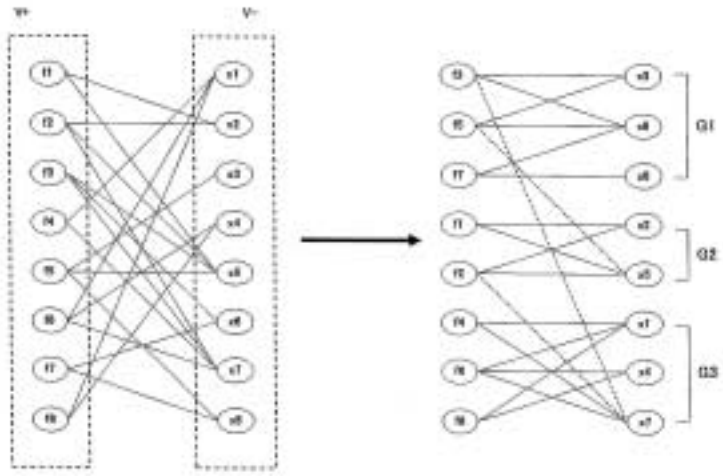

図6：2部グラフをDM分解した結果

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

制約の評価順序指定および変数の優先順位指定により、代数制約評価系の効率的な処理手法について述べる。本手法では、制約評価系に対する制御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ブフバーガアルゴリズムに基づいた代数制約評価に対しては、解制約であるブ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［14］［15］。

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7）。

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

図7：ゴールの並び換えによる制約の評価順序の決定

次に、変数間の優先順位を部分問題G_nの変数集合部分問題G_{n-1}の変数集合≪…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8)、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。

```
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8：述語preによる変数の優先順序の指定

5　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［16］。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9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。

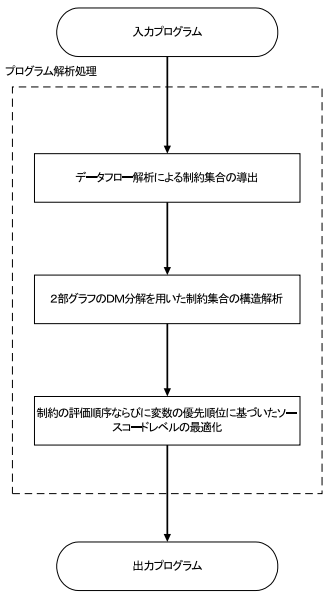

図9：プログラム解析システムの処理概要

5.1 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［10］。

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）［8］［9］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。

なお、この導出処理では、SLD導出に基づいた計算モデル（実行機構）を前提としている。実際の制約論理プログラムの実行は、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、節の決定的な選択を単一化可能な説の逐次実行の選択とバックトラッキングに置き換えたモデル［9］に基づきおこなわれるとみなす。

5.2 DM分解の効率的な計算アルゴリズムへの拡張

図5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）［12］。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 $\{Q_1, Q_2, \cdots, Q_t\}$ を求める。
- ［ステップ2］$M \leftarrow M \oplus Q_1 \oplus Q_2 \oplus \cdots \oplus Q_t$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6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。

5.3 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ブフバーガアルゴリズムに基づいた代数制約評価に対して、解制約であるブ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［14］［15］［16］。

5.4　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8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8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10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6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。

6　実験結果と考察

本章では、例題1を含む線形制約を対象とした7つの問題および例題2を含む非線形制約を対象とした9つの問題に対して、本効率化手法の実験をおこない、それぞれの処理時間を求め、その有効性について考察する。

```
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10：トランスレータにより生成された問題1の解析プログラム

6.1 実験結果

グレブナ基底の計算は制約の評価順序および変数の優先順位に依存する。そこで、本実験では、制約評価系の内部については立ち入らないで、制約評価系を制御し、処理効率を向上させるために、変数の優先順位ならびに制約の評価順序の両者を指定する方法について実験をおこなった。その結果、本方法は表1のように有効性が示されたので、表2および表3の実験に対して適用した。

表1：制約の評価順序および変換の優先順位指定による代数制約処理系の処理時間（単位:msec）

対象とした問題	適用前	適用後
問題 1 (例題 1: 線形, 8 制約, 8 変数)	66	51
問題 2 (線形, 8 制約, 8 変数)	766	688
問題 3 (例題 2: 非線形, 最高次数 3, 8 制約, 8 変数)	610	68

表2は線形制約を対象とした問題について、提案した効率化手法の評価結果を示し、表3は非線形制約を対象とした問題に対する実験結果を示す。実験はSUN4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、制約グラフの構造分解および整合性解析は、SUN4上のSicstus PROLOGバージョン3.0を用いておこなった。

表2：線形制約を対象とした問題の実験結果（単位:msec）

対象とした問題	改善(適用)前	改善(適用)後
問題1 (6 制約, 8 変数)	54	43
問題2 (例題1, 8 制約, 8 変数)	66	51
問題3 (8 制約, 8 変数)	766	688
問題4 (17 制約, 19 変数)	215	185
問題5 (21 制約, 21 変数)	318	278
問題6 (18 制約, 19 変数)	244	227
問題7 (21 制約, 21 変数)	302	258

6.1.1　線形制約の実験結果

線形制約を対象とした場合は、本効率化手法により7パーセントから23パーセントまでの範囲で処理効率が改善された。なお、問題2は前述の例題1の結果を示しており、問題3は制約の数と変数の数が問題2と等しく、定数係数が非常に複雑な分数形式をとる問題である。

6.1.2　非線形制約の実験結果

非線形制約を対象とした場合には、その結果は与えられた問題に依存するが、おおよその場合については表3の適用後1に示されるように処理時間がおおきく改善されており、本効率化手法の有効性が示されたといえる。例えば、問題1は例題2の結果を示しており、約9倍の処理時間の改善がなされている。その場合の実行制御情報は図7の変数の優先順序ならびに図8の制約の評価順序に関する情報を用いた。但し、問題9はその例外であり、制約グラフの構造情報を解析した結果、部分グラフに分解できない、すなわちスパース構造ではなく、密構造なグラフの例である。このような場合には本手法によってわずかしか処理時間が改善されなかったことがわかる。

また、本手法における制約グラフの構造解析においては、変数の共有関係のみを考慮して、構造分解をおこない、最終的には制約の評価順序ならびに変数の優先順位を決定するが、変数の次数に関しても考慮して効率化することが望ましい。そこで、問題7および問題8ではこのような変数の次数に関する情報も考慮して効率化手法を適用した結果について示しており（表3の適用後2）、大幅に効率効率が改善されたことがわかる。たとえば、問題8（制約式の数21、変数の数21、最高次数7）では、本手法を用いないで制約評価をおこなうと約12時間かけても解が得られなかった（∞によりあらわす）が、本手法により数秒で解を求めることができた。

表3：非線形制約を対象とした問題の実験結果（単位:msec）

対象とした問題	適用前	適用後1	適用後2
問題1 (最高次数3, 8 制約, 8 変数)	610	68	–
問題2 (最高次数7, 17 制約, 19 変数)	618	264	–
問題3 (最高次数7, 21 制約, 21 変数)	8554	1778	–
問題4 (最高次数7, 21 制約, 21 変数)	470294	6914	–
問題5 (最高次数7, 21 制約, 21 変数)	1087822	288	–
問題6 (最高次数2, 12 制約, 10 変数)	281279	1322	–
問題7 (最高次数8, 6 制約, 8 変数)	3882	18581	57
問題8 (最高次数7, 21 制約, 21 変数)	∞	∞	2738
問題9 (最高次数2, 11 制約, 11 変数)	2633	2542	–

```
1     E ← F;
2     R ← 0;
3     ReduceAll(E, R);
4     NewBasis(E, R);
5     B ← {{f1, f2} | f1, f2 ∈ E, f1 ≠ f2};
6       while B ≠ 0 do
7         choose a pair {f1, f2} ∈ B;
8         B ← B − {f1, f2};
9         h ← SPolynomial(f1, f2);
10        h' ← NormalForm(h, E ∪ R);
11        if h' ≠ 0 then
12          B ← B ∪ {{e, h'} | e ∈ E};
13          R ← R ∪ {h'};
14          ReduceAll(E, R);
15          NewBasis(E, R);
16        endif
17      endwhile
```

図11：Buchberger アルゴリズムの処理概要

6.2 Buchberger アルゴリズムとその効率化に関する考察

6.2.1 Buchberger アルゴリズム

以下では制約論理型言語CALの代数制約評価系に取り入れられている多項式イデアルのグレブナ基底を求めるBuchbergerアルゴリズムについて示す［14］［15］［17］。

図11にBuchbergerアルゴリズムの主要な処理アルゴリズムを示す。入力である等式集合Eを多項式の有限集合$F=\{f_1,\ldots,f_r\}$、出力であるEをFによって生成されるイデアルのグレブナ基底$G(F)$とする。

まず、$E\leftarrow F$, $R\leftarrow 0$とする。Eのすべての要素について、規則集合Rを用いて簡約化をおこない、規則集合Rを更新する（図13のReduceAll）。その結果から図14のNewBasisによって新しい基底を求める。Eのなかから互いに等しくない2個の要素からなるペア $\{f_1, f_2\}$ をすべて求め、その集合をBとする。

次に、集合Bから任意のひとつの要素$\{f_1, f_2\}$を取りだし、S-多項式hを求め、Bから$\{f_1, f_2\}$を消去する。さらに、Eに関するhの正規表現h'を計算する。もし、$h'=0$でなければ、新たなペア $\{\{g, h'\} \mid g\in G\}$ をBに加え、さらにEにh'を加え、Eのすべての要素について、規則集合Rを用いて簡約化をおこない、基底を更新する。このような操作をBの要素がなくなるまで繰り返す。図15には正規表現を求める正規化アルゴリズムを、図16にはS-多項式を求めるアルゴリズムをそれぞれ示す。

6.2.2 代数制約評価系の効率化に関する検討項目および考察

次に、上記のBuchberger アルゴリズムに基づいた代数制約評価系の効率化について考察する。Buchberger アルゴリズムの効率化については以下の検討項目について考察をおこなった。

［単項の順序付け］

- 変数の優先順位はアルゴリズムの処理時間に非常に影響を与える。

CALでは、変数に対して優先度を指定することにより、次の手順によって単項間の順序を決定する［17］。

1. pre述語によって与えられた優先度に基づいて、変数間の全順序関係≫を求める。

2. 求められた変数間の全順序関係にしたがって、単項を変数の順序が大きいものから並べ、グループ化する。

3. グループ化された単項を比較し、最大の変数を含む単項がいくつか存在するならば、単項どうしを全次数式（total-degree）順序で比較する。もし、等しければそれをもとの単項から取り除いてできる単項に対して上記の処理を繰り返し、どちらか一方が空になったときは、空でないほうの単項を大とする。

Buchbergerアルゴリズムはこのように求められた単項間の順序に基づき、与えられた制約集合の要素である等式（制約）の最大単項をそれ以外の多項式へ書き換えていくことにより、グレブナ基底を求める。pre述語を用いることにより、制約評価系の書き換え規則を指定し、制約の評価を制御できる。

制約グラフの構造情報を解析し、解析された情報に基づいて制約間の依存関係を求め、この依存関係情報からなるべく少ない書き換えにより既約（正規）形が計算されるように、等式の書き換え規則を決定する。図11に示されるBuchbergerアルゴリズムにおいては、3行目のReduceAll、4行目のNewBasis、さらに9行目のSPolynomialや10行目のNormalFormに対してこのような効率化がおこなわれていると考えられる。その結果、制約評価系に対してスパース行列手法を適用し、効率的な制約の書き換えが実現できる。

［生成されるクリティカルペアの数と選択すべき順序］

- 生成されるクリティカルペア $\{f_1, f_2\}$ の数はアルゴリズムの処理時間に非常に影響を与える。

- 集合Bからクリティカルペア $\{f_1, f_2\}$ を選択する順序はアルゴリズムの性能に影響を与える。

- 求められる基底はクリティカルペアからS-多項式を生成する順序に依存する。

図11のBuchbergerアルゴリズムの5行目では、n個の要素の等式集合Eのなかから互いに等しくない2個の要素からなるペア $\{f_1, f_2\}$ を要素とする集合Bを求めており、その要素数は${}_nC_2=n(n-1)/2$である。6行目から16行目ではBの要素がなくなるまで、Bからクリティカルペアを選択し、これよりS-多項式を求め、さらに項の書き換えによってS-多項式の簡約化を繰り返す。この部分ではクリティカルペアの選択は非決定的であり、Bの要素が急激に増加する可能性が考えられる。したがって、その部分を効率的に処理するためには、生成されるクリティカルペアの数をできるだけ少なくすることが必要である。

われわれは制約グラフの構造分解によって、与えられた問題を部分問題に分解し、後述する制約

評価系のもつインクリメンタリティという性質を用いて、部分問題ごとにグレブナ基底を計算し、最終的に全体の基底を求める方法[2)]を選択した。これにより、生成されるクリティカルペアの増加をなるべく抑制し、効率的な処理が可能になると考えられる。

たとえば、例題2を考えてみると、制約の数が8(n=8) であるため、生成されるクリティカルペアの数は${}_8C_2$=8(8−1)/2=28である。一方、提案した効率化手法では部分問題ごとに生成されるクリティカルペアの数は部分問題G_3では${}_3C_2$=3(3−1)/2=3, G_2では${}_8C_2$=2(2−1)/2=1, G_1では${}_3C_2$=3(3−1)/2=3となる。したがって、部分問題間のインタラクション（変数の共有）が少なければ、部分問題ごとに生成されるクリティカルペアの個数（合計数7）に対して、さらにインタラクションのある部分のクリティカルペアの個数を考慮すればよいことになり、前者の${}_8C_2$=8(8−1)/2=28の場合と比較して、その数を減少させることができる。

［制約評価系のインクリメンタリティ］

新しい多項式が基底に追加されるたびに、その多項式を用いて基底中の要素である多項式を簡約化できれば、基底の簡約化が可能である。これにより、基底計算時に生成される基底の数を減らせるので、アルゴリズムの処理効率を向上させることができる。CALやCLP（$\mathcal{R}$）［8］に代表される制約論理型プログラミング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。これは制約論理型言語の推論機構がPrologなどの論理型言語の特徴である操作（手続き）的意味論に基づいた計算モデルであるSLD-導出［18］に制約という概念を拡張したモデルに基づいているためである［8］［9］。以下ではこのように拡張されたSLD-導出の定義を示す。

［定義］Pをプログラム、Rを計算規則、Cを制約、G_iをゴール
$\leftarrow A_1,\ldots,A_k,\ldots,A_m;C$（$1\leq k\leq m$）とする。$D_{i+1}$を$P$内の確定節$B\leftarrow E_1,\ldots,E_n;C_1$（$n\geq 0$）
とし、D_{i+1}のヘッドBとG_iのサブゴールA_kは最汎単一化子（mgu）θ_{i+1}をもつとき、計算規則Rとθを用いて、G_iとD_{i+1}よりゴールG_{i+1}が次の条件で導かれる。

- A_kは計算規則Rにより選択された原子式である。
- $C_{i+1}=C\cup C_1\cup \{B=G_i\}$ が可解（solvable）ならば、G_{i+1}はG_iとD_{i+1}から
  ゴール$\leftarrow (A_1,\ldots,A_{k-1},D_1,\ldots,D_n,A_{k+1},\ldots,A_m,C_{i+1})\ \theta_{i+1}$を導出する。

［定義］Pをプログラム、Gをゴール、Rを計算規則とする。$P\cup\{G\}$ のRによるSLD-導出列とはゴールの列$G_0=G, G_1,\ldots,G_n$とPのプログラム節の変種の列$D_1,D_2,\ldots$と最汎単一化子$\theta_1,\theta_2,\ldots$の列から構成されており、各G_{i+1}はRよりθ_{i+1}を使ってG_iとC_{i+1}から導かれ、nをSLD-導出列の長さという。SLD-導出列がsuccessfulであるとは、その最後の要素が原子式でないゴールをとり、successfulな導出における最後の要素である制約を解制約という。

---

2) CALの実際の制約評価系では、制約の順次追加により制約集合が与えられた場合、あらたな制約が追加されるたびに、最初から解くのではなく、いままでに求められた制約の正規形を修正して解を求めている。これを制約評価系のもつインクリメンタリティという性質ということにする。

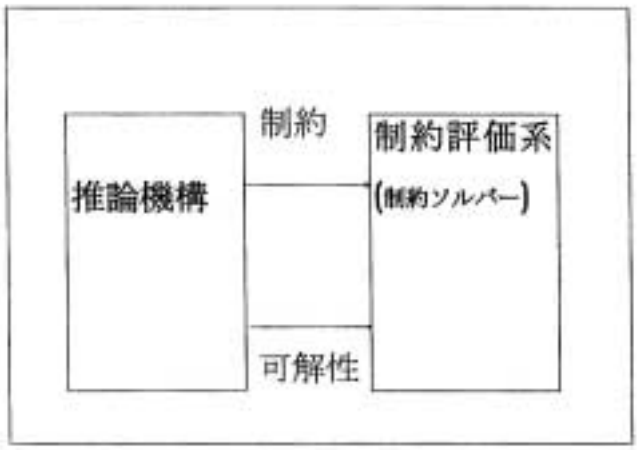

図12：制約論理型言語の推論機構

［定義］Pをプログラムとする。Pの成功集合S_pは次のように定義される。

S_p= ｛(p←A; C)｜ゴール←A; Cが解制約CをもつsuccessfulなSLD-導出列を有する｝

制約は図12のように制約論理型言語の推論機構すなわち、上述の拡張されたSLD-導出に従って評価される。CALプログラムの実行では、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。制約評価系を含む制約論理型言語を用いた処理を効率化するためには、このような制約のバックトラックをなるべく少なくし、冗長な処理を減らすことが必要とされる。そのためには、制約評価系において評価すべき制約集合のもつ代数構造の解析および構造情報の抽出によって、制約間の依存関係情報を生成し、これを用いて無駄な計算（S-多項式の生成）を避け、解の係数成長をできるだけ抑制するように、バックトラックを制御することが考えられる。これは第4章で述べた制約の評価順序を求め、代数制約評価系を効率化する手法で実現されている。

6.3　実験結果に対する考察

6.3.1　線形制約の実験結果

線形制約を対象とした代数制約評価系は解制約であるグレブナ基底を一意に求めることができた。線形制約を対象にした問題では、BuchbergerアルゴリズムではS-多項式は生成されない。これは、線形制約ではクリティカルペアの要素である最大単項が互いに素である（coprime）ため、S-多項式がゼロとなり、S-多項式の計算が不要になるからである。

本効率化手法では、制約の評価順序を指定することにより、無駄な計算をおこなわないで、処理を効率化することができた。特に、図11のBuchbergerアルゴリズムでいうと1行目から4行目までの制約評価の部分がこれに相当し、その主要部である3行目のReduceAll（E,R）と4行目のNewBasis（E,R）の処理が効率化されたと考えられる。しかしながら、その処理効率に非常に影響を与えるS-多項式に関する計算がないため、非線形制約の問題ほど急激には処理時間は改善されなかった。

6.3.2　非線形制約の実験結果に対する考察

非線形制約を対象とした代数制約評価系では、特に、クリティカルペア $\{f_1,f_2\}$ に対するS-多項式の計算がBuchbergerアルゴリズムにおけるグレブナ基底の計算時間に大きな影響を与える。本手法では、ブロック三角化手法によって問題を部分問題に分解し、それぞれの部分問題に対して、順

番にグレブナ基底を解として求めていく。このような三角化手法の適用によって、クリティカルペアに関する計算を減少させることができる。

その理由は、与えられた問題全体に対してクリティカルペアに関して計算するのではなく、分解された部分問題に対応する多項式の部分集合に対してのみ計算するためである。つまり、問題を部分問題に分割することによって、生成されるS-多項式を減少でき、依存関係情報を用いることにより集合Bから冗長なクリティカルペアをできるだけ取り除けるので、クリティカルペアに関する計算を減少できるためである。さらに、基底計算において部分問題に対して求められているグレブナ基底を規則集合Rとみなし、分割された部分問題間の相互依存関係がなるべく少なくできれば、クリティカルペアに関する計算、つまり生成されるS-多項式の数の減少ならびに解の係数成長の抑制により、効率的な処理が実現できると考えられる。

6.4 問題点と今後の課題について

本手法では制約集合があらわす制約グラフのスパース性（疎行列）という代数的特徴を用いて、処理を効率化するものであるが、次の項目については十分対処されておらず、今後の課題として検討していく予定である。

- 本制約評価の効率化手法に適した問題についての再検討
- 変数の共有関係だけでなく、共有する変数の次数に関しても考慮する必要がある。
- 部分問題を評価する（解く）順序は部分問題間の半順序関係を用いて決定している。しかしながら、この順序の決定方式では不十分であり、クリティカルペアの増加を抑え、グレブナ基底の係数をできるだけ成長させないような方式を考えることが必要である。
- 本効率化手法と代数制約評価（ブフバーガアルゴリズム）との関係についての再考察
- 制約論理型言語のふるまいについてのより詳しい解析のために必要となる抽象解釈ならびにfold/unfoldなどのプログラム変換技法の検討
- 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

7 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、プログラム解析でおこなわれている解析手法ならびにグラフ論的な手法の適用により制約論理型言語において代数制約評価系を効率化する手法について検討した。

次に、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。

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. Mannita and E. Ukkonen: Flow Analysis of Prolog Programs, *Proc. of SLP '87,*（1987）.

［4］K. Marriott and and H. Sondergaard: Analysis of Constraint Logic Programs, *Proc. of NACLP '90,*（1990）.

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

［6］J. Cohen: Constraint Logic Programming Languages, *Comm. of the ACM,* Vol.33, No.7,（1990）.

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

［8］J.W. Lloyd: *Foundations of Logic Programming,*（邦訳：論理プログラミングの基礎、佐藤雅彦他訳）, Springer-Verlag,（1984）.

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

［10］M.S. Hecht: *Flow Analysis of Computer Programs,* North-Holland,（1977）.

［11］K. Marriott and H. Sondergaard: Analysis of Constraint Logic Programs, *Proc. of NACLP '90,*（1990）.

［12］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）.

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

［14］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）.

［15］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）.

［16］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）.

［17］永井保夫、長谷川隆三：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、日本ソフトウエア科学会第8回大会、B2-2,（1991）.

付録

［すべての等式（多項式）の簡約化］

等式集合Eのすべての要素に対して、規則集合Rを用いて簡約化をおこない、既約な等式をあらたな規則集合Rに追加する。

［基底の更新］

```
ReduceAll(E, R);
1    while E ≠ 0 do
2      choose an element h ∈ E;
3      E ← E − {h};
4      h ← NormalForm(h, E ∪ R);
5      if h ≠ 0 then
6        R ← R ∪ {h};
7      endif
8    endwhile
```

図13：等式の簡約化アルゴリズム

入力である等式集合EとReduceAllにおいて求められた規則集合Rを用いて、新しい基底集合を求める。

```
NewBasis(E, R);
1    E ← E ∪ R;
2    H ← E;
3    K ← 0;
4      while H ≠ 0 do
5        choose an element h ∈ H;
6        H ← H − {h};
7        k ← NormalForm(h, E − {h});
8        K ← K ∪ {k};
9      endwhile
10   E ← K;
```

図14：基底の更新アルゴリズム

[正規化]

正規化アルゴリズムは多項式gおよび多項式集合$F=\{f_1,\ldots,f_n\}$が与えられたとき、Fの要素である多項式を用いてgを書き換え、これが既約になるまで簡約化を繰り返し、正規化表現を求める。

```
NormalForm(g, F);
1    g_0 ← g;
2    i ← 0;
3    while exist f ∈ F such that 最大単項 (leading power product)
     of f divides 項 (power product) p in g_i
4        g_{i+1} ← g_i − b̂ · u · f_i;
5        i ← i + 1;
         where b̂ は多項式 p を多項式 f の最大単項の係数で除算した商
               u= p/ 多項式 f の最大単項
6    endwhile
```

図15：正規化アルゴリズム

[S-多項式の定義]

正規化アルゴリズムでは、多項式集合$F=\{f_1, f_2\}$の要素である多項式の全次数がgのそれより大きい場合には書き換えおよび簡約化ができない。その解決策として、多項式集合の生成元を追加し、正規化アルゴリズムを適用することが考えられた。この生成元を多項式f_1とf_2のS-多項式と呼び、定義は以下に与えられる。

```
SPolynomial(f_1, f_2)
1    u_1 · f_1 − (c_1/c_2) · u_2 · f_2;
2    where c_i = 多項式 f_i の最大単項の係数,
3          u_i is such that s_i · u_i = s_1 と s_2 の最小公倍数
4            and
5          s_i = 多項式 f_i の最大単項 (i= 1,2)
```

図16：S-多項式の定義