記 号 処 理 47－1
（1988. 9. 22）

制約ロジック・プログラミング言語CAL
－開発の現状と構想－
Constraint Logic Programming Language CAL
– Its Development and Future Extensions –

坂井 公　相場 亮
Kô Sakai and Akira Aiba
(財) 新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

4-28, Mita 1-Chome, Minatoku, Tokyo 108 JAPAN

概要

本論文においては、現在ICOTにおいて開発中の制約論理型言語CALの開発の現状について述べる。また、将来の拡張構想について述べる。CALは現在開発中の言語であり、現時点においては、非線形のものを含む代数等式、あるいはブール等式を制約として記述、評価できる言語として、PSI上に実装されている。将来的には、複数の制約評価系をひとつのシステムに組み込むことによって、柔軟な制約ロジック・プログラミングの機構の提供を目差している。

ABSTRACT

We report the current stage of the development and future extensions of constraint logic programming language CAL which is being developed at ICOT. At present, CAL is implemented on PSI. It can handle constraints in the form of algebraic equations including non-linear ones, and boolean equations. Both are handled by the critical-pair completion method. In future, we will present a flexible programming environment for constraint programming by providing a plurality of constraint solvers in the system.

1　はじめに

CALは現在ICOTにおいて研究・開発中である制約ロジック・プログラミング言語である。これまでの制約ロジック・プログラミング言語と比較してCALは(1)現在実装されている制約評価系の能力が他の処理系には見られないこと、(2)ロジック・プログラミング言語を用いて実装を行なっているおかげで制約評価系との組み合せが柔軟であることに、その特徴がある。本論文では、CALの開発の現状と将来の構想について述べる。なお、CALの意味論的側面については、[SaA-88]を参照されたい。

制約プログラミングは、さまざまな分野において応用が期待されている新しいプログラミング・パラダイムである。その特長は宣言的であるということであり、プログラマは目標を設定するだけで、その目標を達成する方法については、原則的には触れずに問題を解くところにある。

計算機を用いてある問題を解決しようとする場合、まず問題を厳密に記述することが必要である。そのためには、問題領域とその領域における対象とを設定しなければならない。コンピュータ・グラフィックスでいえば、たとえば2次元空間が領域であり、そこにある各種図形が対象ということになる。数値計算における数値定数や変数なども対象であるといえる。

ある問題領域における対象の集まりは、通常、ある系をなす。たとえば、複数の質点の運動を考えると、これらは互いに干渉しあうような、ひとつの物理系を形成する。質点の衝突は、同一座標に複数の質点が同時刻に存在することであり、これはたとえば平面内の座標x、yと時刻tの関数で表わされる質点同士の関係で表現される。

制約とは、このように対象間の関係を宣言的に記述したものである。制約の形で問題を記述し、それを解決するというパラダイムをロジック・プログラミング言語に組み込んだものが制約ロジック・プログラミング言語である。

本論文においては、第2節でCLP($\mathcal{R}$)を例にとって制約ロジック・プログラミング言語の紹介を行なう。第3節ではCALの現状について述べる。第4節でCALの拡張構想について述べる。

2　制約ロジック・プログラミング言語

最初に提唱された制約ロジック・プログラミング言語はColmerauerの　Prolog-II [Col-82]である。その後

Jaffar 、 Lassez 等によって制約ロジック・プログラミング言語スキーマ CLP(**X**) [JaL-86] が提唱され、制約ロジック・プログラミング言語の意味論が明確化された。この言語スキーマは、その特殊な場合として Prolog 、Prolog-II 、 Prolog-III [Col-87] 、あるいは CLP(**X**) のインスタンスとしてオーストラリアで開発された CLP($\mathcal{R}$) [JaM-85] などを含む。この観点から見ると、それぞれの制約ロジック・プログラミング言語の差異は、その言語によって記述され、解かれる制約の種類および範囲にある。たとえば Prolog-III で記述可能な制約は、線形方程式、ブール等式であり、 CLP($\mathcal{R}$) では代数方程式および不等式である。ただし CLP($\mathcal{R}$) で解くことのできる制約は線形のものに限られる。

これら以外にも Dincbas による言語 [Din-87] などが制約ロジック・プログラミング言語といえる。

このような制約ロジック・プログラミング言語はいずれも制約評価系 (*Constraint Solver*) と呼ばれる機構をロジック・プログラミング言語の処理系に外付けしているか、あるいはユニフィケーションを拡張することによって対処している。これらのうち、ユニフィケーションの拡張という方法で扱うことのできる制約、すなわち関係は等式のみであるため、より一般的な議論を行なうために制約評価系の外付けによる対処を例にとって制約ロジック・プログラミング言語の処理を概説する。

制約は「制約記号」、たとえば「等号」や「不等号」によって他の述語呼び出しと区別される。プログラムは Prolog のプログラムと同様に評価されていくが、制約記号を持つリテラルに出会うと述語を呼び出すかわりに制約評価系と呼ばれるシステムが起動される。

制約評価系は、それまでに保持していた制約の集合、より正確にはその標準形に新たに得られた制約を付加し、新しい標準形を求める。ただし、新たに得られた制約がそれまでの制約集合に対して矛盾する場合には、制約評価自体が失敗し、プログラムに応じてバックトラック等が生じる。

制約ロジック・プログラミング言語においては制約評価系として何を採用するか、採用にあたってはどのような選択基準を設けるべきなのかということが問題になる。これは言い換えればどのような計算領域のどのような関係を制約として記述・評価するべきかという問題になる。 Dincbas は、どのような領域の等式を制約としてユニフィケーション中に組み込めば良いかという問題について次のような基準をあげている [Din-87] 。

1. その領域の各等式は完全性を失なわずに決定的に解けること。
2. その領域において効率の良い等式の解法が存在すること。
3. その計算領域はユニフィケーションの中に組み込むことが正当化できるほど頻繁に用いられること。

われわれはこれに対し、等式に限らず制約として導入すべき領域について次のような基準を提唱した [SaA-88]。これらの内 1. から 3. は上の基準に準拠したものである。

1. 制約の充足可能性 (解を持つかどうか、可解性ともいう) が決定できること。
2. 充足可能性を決定する効率の良いアルゴリズムがあること。
3. その計算領域は制約の領域として導入することが正当化できるほど頻繁に用いられること。
4. 充足可能な制約についてはある種の標準形が計算でき、それがその制約の解と同一視できること。

4. は、たとえば、 `X+Y=1, X-Y=1` と `X=1, Y=0` とは同等な制約であるが、このとき後者を標準形とすれば前者から後者が計算できるという意味である。

3 CAL

3.1 CAL 処理系の現状

現状の CAL は単一の言語ではなく、 CAL という名称のもとに複数の領域における制約が記述できる複数の処理系が作成されている。現状において利用可能なものは以下の通りである。

DEC2060 においては代数方程式を制約として記述・評価できる処理系 (以下 代数 CAL 処理系 と呼ぶ)、ブール等式を制約として記述・評価できる処理系 (以下 ブール CAL処理系 と呼ぶ) があり、一方 PSI においては、代数 CAL 処理系、ブール CAL 処理系、および両者を共用できる処理系 (以下 型付き CAL 処理系と呼ぶ) がある。

実際の CAL 処理系においては、第 3 節で述べたような意味論に厳密に従った形でプログラムの評価がなされるのではなく、ちょうど Prolog の意味論と実際の処理系との関係に対応するように、意味論には従わない、あるいは意味論を破壊するような機能を、プログラマの便宜のため、付加している。以下では実際の処理系について述べている関係上、このような機能についても触れる。これらについては、そのつど述べることにする。

CAL 処理系は、DEC2060 、 PSI を問わず、以下に示すような構成を持つ。

1. プリ・プロセッサ

   プログラム、ゴールを DEC2060 の場合には DEC-10 Prolog に、 PSI の場合には ESP に変換する。

2. 制約評価系

   制約を受け取り、それを標準形へと評価する。

3.2 CAL プログラムの評価

CAL 処理系は「代数 CAL 」、「ブール CAL 」、「型付き CAL 」を問わず DEC2060 上では DEC-10 Prolog に、PSI 上では ESP へと変換され、それぞれ DEC-10 Prolog 、および ESP のプログラムとして評価される。以下においては DEC2060 の場合を例にとって評価の様子を概観する。

通常の Prolog プログラムの評価と制約ロジック・プログラムの評価の最大の相違は、前者はユニフィケーションの結果を保持しなければならないのに対し、後者は制約の評価結果を保持しなければならないという点にある。すなわち制約ロジック・プログラミングにおいては制約も一種のリテラルであるから、制約の評価結果はリテラルの評価順に従って動的に保持されなければならない。CAL 処理系においては、これを前の評価結果用の入力変数と、新たな評価結果用の出力変数を付加するという、DCG (*Definite Clause Grammer*) と同様の手法により、管理している。

CAL のプログラム

```
zmult(c(R1,I1),c(R2,I2),c(R3,I3)) :-
        R3 = R1*R2-I1*I2,
        I3 = R1*I2+R2*I1.
```

は次のような Prolog プログラムへと変換される。

```
zmult(c(R1,I1),c(R2,I2),c(R3,I3),V0,V1) :-
        constraint([R3 = R1*R2-I1*I2,
                    I3 = R1*I2+R2*I1],V0,V1).
```

この中で 3 引数の述語 constraint は第 1 引数として与えられた制約を第 2 引数として与えられた標準形をしている制約の集合に付け加え、新たな標準形を求め、これを第 3 引数にユニファイするということを行なう。第 2 引数の初期値は制約がまったく無い状態であり、一般には制約評価系毎に定められる。代数 CAL の場合、空リストである。

したがって Prolog の組込述語を利用する場合を除いて新たに 2 つの引数を付け加えるという作業を行なう必要がある。これを行なうのがプリ・プロセッサである。

このようにして変換されたプログラムは、もはや Prolog のプログラムとなっており、したがってこのプログラムを評価するためには上に書いたような仕様を充たす述語 constraint を用意すれば良いということになる。

以上が CAL 処理系におけるプログラムの評価の概要である。

3.3 制約評価のアルゴリズム

前節の例からも分かるように制約はインクリメンタルに解かれることが要求される。すなわち制約は同時に得られるのではなく、一つづつ順次得られるため標準形は常に以前の標準形と新たに得られた制約とを変形することによって求められることになる。

代数 CAL の特長は代数方程式、すなわち代数的等式であれば非線形のものでもある意味での解を求められるという点にある。代数 CAL では数式処理や図形定理証明系等の分野において近年盛んに利用されている、多項式の Gröbner 基底を求めるための Buchberger アルゴリズムを制約評価アルゴリズムとして採用している。

一方ブール CAL においてはこの Buchberger アルゴリズムを元に考案された、Boolean Gröbner 基底を求めるアルゴリズム [SaS-88] を制約評価アルゴリズムに採用している。

以下これらについて概説する。

3.3.1 Gröbner 基底

この節においては代数 CAL において用いられている制約評価アルゴリズムについて述べる。なお詳細については文献 [SaA-88] を参照されたい。

Buchberger は Gröbner 基底の概念を導入し、与えられた多項式系の Gröbner 基底を求めるアルゴリズムを示した [Buc-83] 。

一般性を失わずに、多項方程式が $p = 0$ という形をしていると仮定することができる。$E = \{p_1 = 0, p_2 = 0, \ldots, p_n = 0\}$ を多項方程式系とする。I を $\{p_1, p_2, \ldots, p_n\}$ から生成される多項式環のイデアルとする。元素と E の解の間には次のような関係がある。

定理 3.1 (Hilbert の零点定理)
p を多項式とする。E のすべての解が $p = 0$ の解でもあるのは、ある自然数 n に対して p^n が I の元であるとき、かつそのときに限る [Hil-90]。

また次の系は制約の可解性の判定において重要な意味を持つ。

系 3.1
E は、$1 \in I$ であるとき、かつそのときに限り解を持たない。

以上によって多項方程式 $p = 0$ を制約 E の元で評価するという問題は多項式 p^n が E から生成されたイデアルに属するか否かを判定する問題に帰着された。ここで注意しなければならないことは制約とイデアルとの関係が完全ではないことである。たとえば $X = 0$ は制約 $\{X^2 = 0\}$ のもとで成り立つが、$\{X^2 = 0\}$ から生成されるイデアルには属さない。

Buchberger は多項式がイデアルに属するか否かを決定するアルゴリズムを与えた。E を与えられた等式の集合、R を書換え規則の集合とする。

多項方程式系において各方程式は単項間にある順序を仮定した場合に、その順序のもとでの最大の単項をそれ以外の多項式に書換えるような書換え規則であると考えられる。任意の 2 つの書換え規則の左辺が互いに素では

ない場合、それら左辺の最小公倍数は2つの規則によって2つの多項式に書換えられる。これをそれらの規則の要対 (*critical pair*) と呼ぶ。要対の中には、それらをさらに規則によって書換えていってもその結果が異なる、すなわち書換えが合流しない場合がある。このような要対のことを発散する要対と呼ぶ。E を与えられた等式の集合、R を書換え規則の集合とする。

1. $R \leftarrow \emptyset$
2. E 中の等式 $l = r$ のすべてについて、$l - r$ を R 中の書換え規則と算術演算とによって単純化し、式 e を得る。$e \equiv 0$ であれば、この等式を捨てる。そうでなければ、E 中の等式 $l = r$ を $e = 0$ で置き換える。
3. $E = \emptyset$ であれば終了。
4. E 中から等式 $e = 0$ を選ぶ。
5. e 中の、ある順序に従った場合の頭項を l' とする。$e = 0$ を l' について解き、式 $l' = r'$ を得る。
6. 規則 $l' \rightarrow r'$ を R に付け加える。
7. R 中の規則の発散する要対を等式として E に付け加える。
8. 2. へ行く。

次の定理はイデアルと Gröbner 基底との関係を述べたものである。

定理 3.2 (Buchberger)
R を方程式系 $\{p_1 = 0, p_2 = 0, \ldots, p_n = 0\}$ の Gröbner 基底、I を $\{p_1, p_2, \ldots, p_n\}$ から生成されるイデアルとする。このとき多項式 p は R によって 0 に書換えられるとき、かつそのときに限り I に属する。

また次の定理は Gröbner 基底をもって制約の標準形と考えることの妥当性を保証するものである。ここで既約な Gröbner 基底とは、その中の2つの書換え規則が互いに他を書換えることがないもののことである。

定理 3.3
単項間にある順序が固定されているとする。E、F を方程式系とする。このとき E、F から生成されるイデアルが等しければ E、F の既約な Gröbner 基底は等しい。

実際に代数 CAL の制約として記述できるものは制約記号として、"=" と "==" を持つものである。$f = g$ はいわば能動的な (本論文を通じて論じている本来の意味での) 制約で、それを加えて新たな標準形を求める。すなわち、$f = g$ が成立するように Gröbner 基底を変形することを意味する。一方、$f == g$ は受動的な制約で、$f = g$ が現在までに集められた制約のもとで成立するかどうかを調べる。ここで受動的というのは Dincbas の意味とは異なり、制約の標準形を変更することがないという意味である。これは、Prolog で "==" がチェックだけでユニフィケーションを引き起さないのと事情は同じである。先に注意したように、制約とイデアルとの関係が完全ではないので、Gröbner 基底による単純な書換えだけではこのチェックを行なうことはできない。しかし、Gröbner 基底の方法を巧妙に利用することによって判定が可能である。

この "==" は、利用者の便宜のため、付け加えたものである。

3.3.2 Boolean Gröbner 基底

この節においてはブール CAL の制約評価系で用いられているアルゴリズムについて述べる。ブール CAL はブール等式を制約として記述・評価できるものであり、典型的な領域としては真偽値の集合があげられる。制約評価系においては Boolean Gröbner 基底を求めるための、もとの Buchberger アルゴリズムとは少し異なったアルゴリズムを用いている [SaS-88]。

ブール方程式の可解性の判定法は多く知られているが、その中の典型的なものは意味ユニフィケーション (*semantic unification*) によるものであろう。この方法は Dincbas [Din-87] が採用している。

一方 CAL において採用したのは一般の多項式と同様に Gröbner 基底によるアプローチである。このアプローチは意味ユニフィケーションによるものと比較して以下の点で優れていると考えている。

1. プログラム、またはゴールに書かれた以外の変数をシステムが導入するということがないため、出力が分かりやすい。
2. 制約の標準形が存在し、その意味も明確である。

3.4 プログラム例

本節においては、これまでに述べてきた CAL の各制約評価系に関しその例を示すことによって CAL の記述能力を紹介する。

3.4.1 代数 CAL のプログラム例

前述したように代数 CAL の特長は非線形方程式を解くという点にあるが、まず線形の例をあげる。以下に示す例は問題としては単純な鶴亀算であるが、これがさまざまなゴール列に対してさまざまな結果を出力するという点に CAL の特徴が表われている。

例 3.1 (鶴亀算)
鶴亀算を解くプログラムを CAL で記述すると次のようになる。

```
trkm(Tsuru,Kame, Ashi, Atama) :-
            Atama = Tsuru+Kame,
            Asi = Tsuru*2+Kame*4.
```

これに対して以下のようなゴール列を評価させる。

1. `?- trkm(2,3,legs,heads).`

   鶴の数、亀の数が既知で頭の数、足の数を求める場合で単純な計算によって解くことができる。この場合の解は `legs = 16, heads = 5` となる。

2. `?- trkm(crane,tortoise,16,5).`

   いわゆる鶴亀算の場合で連立方程式を解く必要がある。この場合の解は `crane = 2, tortoise = 3` となる。

3. `?- trkm(crane, 3, legs, heads).`

   未知数の値を決定するには方程式の本数が足りない。このような場合 CAL ではこれら未知数の間の関係式が出力される。この場合の出力は `heads = 3+crane, legs = 12+2*crane` である。

4. `?- trkm(2,3,14,heads).`

   この場合には 制約評価は失敗し、「no」という結果が出力される。

次に例として示すのはラグランジュの未定乗数法を用いて条件付き極値を求める問題である。

例 3.2 (条件付き極値)
ラグランジュの未定乗数法を CAL で記述し、それを用いて条件付き極値を求めた例を示す。

ラグランジュの未定乗数法を CAL で記述すると、次のようになる。

```
ex(F, Constraint,Vars) :-
                lag(Constraint, Lag),
                difs(Vars, F, Lag).
lag([ ], 0) :- !.
lag([L=R |Cs], Mult*(L-R)+Lag) :-
                L=R,
                lag(Cs, Lag), !.
difs([ ], _, _) :- !.
difs([Var |Vars], F, Lag) :-
                dif(F, Var)=dif(Lag, Var), !,
                difs(Vars, F, Lag).
```

述語 ex の第1引数は極値を求める目的関数であり第2引数はその際に考慮される条件、そして第3引数はそれらの式のうち値を変更しうるもの(すなわち変数)のリストである。プログラム中の dif(F,Var) は、多項式 F を変数 Var で微分して得られる多項式を表わしている。書き下しても数行のプログラムであるが、プログラマの便宜のため、 CAL では組み込みになっている。

厳密に CAL の意味論に則して考えると、dif(F,Var) は Prolog で言う項であって、多項式ではないので、制約として記述するには問題があるのであるが、利用者の便宜のためにあえて導入したものである。

このプログラムを用いて次のような問題を解くことを考える。

問題 3.1 (数学セミナー 1985 年 9 月号)
いま半径が *1* の円があるとする。この円を *2* 本の半径で *2* つの扇型に分割する。それぞれの扇型から円錐を作る。このとき *2* つの円錐の体積の和を最大にするには、円をどのように分割すれば良いか。

次のように考える。まず円錐の体積の和は扇型の中心角が π の点に対して対称である。したがって2つの扇型の中心角をそれぞれ $1/2+r$ と $1/2-r$ とすることにする。次に中心角 $1/2-r$ の扇型から作った円錐の高さを sB 、もうひとつの扇型から作った円錐の高さを sA とする。

このとき、この問題は次の呼び出しによって解くことができる。

```
ex((1/2+r)^2*sA+(1/2-r)^2*sB,
   sA^2+(1/2+r)^2 = 1, sB^2+(1/2-r)^2 = 1],
   [sA, sB]).
```

このプログラムは3つの規則からなる Gröbner 基底を出力する。この基底には次のような r のみからなる式が含まれている。これは r の 7 次式であるが、両辺を r で割れば、r^2 の 3 次式となるので根を求めることができる。この根は題意の体積の和の極値を与える値であり、これらの中に求める値が含まれる。

```
r^7 = (29/12)*r^5+(-17/48)*r^3+(5/576)*r
```

3.4.2 ブール CAL のプログラム例

前述したようにブール CAL において解くことのできる制約はブール等式である。この点でプログラム例として考えられるものは論理回路の検証である。

例 3.3 (平面交差回路)
この例は [Din-87] において用いられているものである。問題はこの回路が(平面)交差回路であることを検証することである。

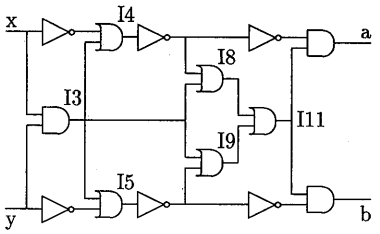

図 -1: 平面交差回路

この回路が交差回路であることをブール CAL を用いて検証するには次のようなことを行なえば良い。まず回路の仕様をブール式を用いて記述する。その上で入力端子と出力端子との関係を求めれば良い。そのプログラムを次に示す。

```
cir(X,Y,A,B) :-
      I4 = ∼XVI3, I3 = XΛY, I5 = ∼YVI3,
      I8 = ∼I4VI3, I9 = ∼I5ΛI3,
      A = I4ΛI11, I11 = I8VI9, B = I5ΛI11.
```

このようなプログラムに対して次のようなゴール列を評価する。

```
?- cir(x,y,a,b).
```

するとシステムは次を出力する。これがこの回路が交差回路であることを示している。

```
a=y

b=x
```

4 CAL の拡張構想

これまで見てきたように現在 CAL には 3 つの制約評価系が存在する。さらに PSI 上には代数 CAL とブール CAL の機能を同時に使えるような型付き CAL を実現した。本節では型付き CAL とその発展形として考えている将来の拡張について述べる。

4.1 線形等式・線形不等式のための CAL

線形等式・線形不等式を扱う制約ロジック・プログラミングの制約評価系には、単体法に基づくものや、Min-Max 法によるものなどが提案されている。われわれは先の基準に照らして、単体法を行なっていく制約評価系を試作中である。しかし、現在の形の標準形は、そのままでは人間にとって分かりやすいものとはいえないので、

- 解制約の出力方法
- 制約の標準形
- 制約評価アルゴリズム

のすべてについて、さらに検討が必要であると考えている。

4.2 型付き CAL

型付き CAL の基本的発想としては複数の制約評価系を同時に使えるようにするということにあった。これを可能にするためには、ある制約を記述する際に、その制約がどの制約評価系で評価されるべきであるのかを指定する方法が必要であった。このために導入したのが各制約に対する型である。型付き CAL を考えた時点で利用可能な制約評価系は Gröbner 基底を求める Buchberger アルゴリズム、Boolean Gröbner 基底を求めるアルゴリズムであった。現在この 2 つの制約評価系を持った型付き CAL が PSI 上に実験的に実装されている。さらに拡張構想として線形不等式に対する制約評価アルゴリズムと、実閉体における制約評価アルゴリズムを実装の予定である。したがって、これら 4 種類の制約評価系に対応する型を制約式に対して付加する。この付加によって、処理系はどの制約評価アルゴリズムを呼び出せば良いかを認識できるわけである。

制約に対する型は次のように付加する。型として許すものは、 alg (algebraic number – Gröbner 基底を求めるアルゴリズムを呼び出すためのもの)、bool (Boolean Gröbner 基底を求めるアルゴリズムを呼び出すためのもの)、lin (linear – 線形不等式のための制約評価アルゴリズムを呼び出すためのもの)、および real (実閉体の決定手続きの部分アルゴリズムを呼び出すためのもの) である。これらを制約式の後ろに [illegible]
より型を指定する。

さらに各節頭部の引数に式を書くことを許すため述語に対する型宣言を導入する。たとえば :- type p(alg, bool, _). という型宣言があったとする。この宣言は述語 p の第 1 引数は Gröbner 基底を求めるための代数制約であり、第 2 引数はブール制約であり、第 3 引数は通常のロジック・プログラミングにおける引数であることを表わしている。したがってこの述語 p の定義において、たとえば次のようなことを記述することが可能である。

```
p(X+Y, ∼ B, C) :- Q .
```

ここで、この述語定義に対して次のような呼び出しを行なったとする。

```
p(5, true, a).
```

このとき、これらの間のユニフィケーションは成功して制約として X+Y=5 という代数制約と ∼ B = true というブール制約が得られ、{C/a} という束縛が得られる。これは次による。

前掲の述語定義がシステムに対して入力されるとこの節はプリ・プロセッサによって次のような節へと変換される。

```
p(D1, D2, C) :- X+Y=D1:alg, ∼ B = D2:bool, Q
```

したがって上記のような制約および束縛が得られる。

この述語に対する型宣言を導入することによって Dincbas [Din-87] のいう意味ユニフィケーションとほぼ同等の機能を実現することができる。

型宣言を用いた例を示す。次に示す節は鶴亀算を解くためのものであり制約に対する型付けのみを用いて型宣言を用いずに記述したものである。

```
mah(Man,Horse, Legs, Heads) :-
        Heads = Man+Horse:alg,
        Legs = Man*2+Horse*4:alg.
```

述語に対する型宣言を導入することによりこのプログラムは次のような単位節の形式で記述することが可能となる。

```
:- type mah(_, _, alg, alg).
mah(Man, Horse, Man*2+Horse*4, Man+Horse).
```

5 結論

CAL は現在開発中の言語であり、最終的な姿がどうなるかについては、現時点では軽々に判断することはできない。CAL の今後の発展としては以下の 3 通りが考えられよう。

1. 新たなる制約評価系の実装

   たとえば上に述べたような線形不等式論に基づく線形不等式制約のための制約評価系や、限量子除去アルゴリズムの部分アルゴリズムの実装などが考えられる。

2. アプリケーションに向けた、制約評価系の複合化

   現在 Gröbner 基底の応用として、めざましい進歩をとげているものに幾何の定理証明系がある。CAL のひとつの応用としてこれをとりあげることを考えているが、そのためには Gröbner 基底を求める制約評価系だけでは不足である。上掲の限量子除去アルゴリズムの部分アルゴリズム等の実装と、それに加えてこれら複数の制約評価系が単一の環境として使えなければならない。このためのキーワードが型付き CAL になるはずである。また一般に、より汎用のアルゴリズムは特化されたものと比較して効率が悪いということを考えると制約評価系の階層化ということも考えられる。

3. 並列版 CAL

   並列版制約ロジック・プログラミング言語がどのようなものになるかということは現時点では答えるのが難しい問題であるが、将来的には並列化の方向へ進むことになろう。

CAL の意味論 [SaA-88] は、主に Jaffar と Lassez [JaL-87] に従ったものであるが、若干の相違がある。われわれは制約記号と述語記号とを分けて考えた。一般に CAL でプログラムを書こうというプログラマにとって述語記号によって表現された関係が何を意味するかということも、どうなっていればその関係が満たされるのかも既知のことでなければならないはずである。なぜなら、その記号を導入したのはプログラマ自身だからであり、一方、制約記号については、プログラムを書くためには、その意味に関しては知っている必要があるが、システムがどうやって制約をあつかうのかに関しては知る必要がない。この意味で制約記号は CAL に組み込みであると考えている。したがって制約記号と関数記号の意味は構造としてあらかじめ与えられたものと考え、述語記号の意味はプログラマによって与えられるものであると考えなければならない。このようにあらかじめ記号を分けて考えることによって意味論を自然に定義することができるのである。[JaL-87] においては制約は節において他のリテラルよりも先行するものとしてあつかわれている。われわれは一般性を保つ観点から、このような仮定は導入しなかった。またわれわれは definability や solution compactness 、あるいは satisfaction completeness について論じなかった。これは、制約ロジック・プログラミングでの negation as failure についてわれわれにさほど関心がないためである。negation as failure に適合しないような述語は数多くある。また仮にある述語がこのような否定に対して適しているとしても、その述語に対しては決定手続きがあることが多い。そのような決定手続きは制約ロジック・プログラミングの制約評価系に組み込む方がより自然であると考えたからである。かわりに、われわれは制約の標準形について論じた。それがシステムからの出力として適当なものであると考えたからである。

CAL においては、引数間の関係を結果として出力することがしばしばある。特にゴール中の多くの引数がフリーのまま残されることがある。これは部分評価 [Tak-86] の効果と非常に似たものであり、またロジック・プログラミングにおける unfolding の技法 (たとえば [Tam-84]) にも近いものである。むしろ通常のロジック・プログラミングより CAL における結果の方がより印象的であり、また効果的でもある。これは Gröbner 基底の計算の方がユニフィケーションよりも複雑であるためであり、部分計算や unfolding のような技法は、制約ロジック・プログラミングにおいてその効果を真に発揮すると考えられる。

代数 CAL の現バージョンにおいては制約中の各変数の (仮想的な) 値は代数的数、すなわち整数係数多項方程式の解になりうる複素数である。しかしながら、ある変数 x が、たとえば実数しかとりえないということが分かっていれば $x^2 + 1 = 0$ のような制約があれば、ただちに矛盾が導かれる。したがって、より制限された領域 (上の場合は実数) についての知識を持った強力な制約評価系が

あれば、ある種の問題の計算時間は劇的に減少する。また、さらに $\sin(x) = 1$ や $e^x = \pi$ といった、代数的ではない制約を書きたいということもあるだろう。このような場合には領域をすべての複素数からなる集合へと拡張する必要があるかもしれない。

制約を書き、あるいは解くにあたっては、これら以外にもさまざまな要求が考えられる。このような要求を満すには制約評価系が利用者によって完全に変更可能になっていなければならない。この方針にしたがって、システムは利用者自身の目的のために制約評価系を定義できるように設計された。制約評価系の提供者がしなければならないことは、制約の意味を厳密に定め、先の基準を満足する制約評価系をインプリメントすることである。

[参考文献]

[Buc-83] B. Buchberger, "Gröbner Bases: an Algebraic method in Polynomial Ideal Theory", Technical Report, CAMP-LINZ, 1983.

[Col-82] A. Colmerauer, "PROLOG-II – Reference Manual and Theoretical Model", Internal Report, Groupe Intelligence Artificille, Universite Aix-Marseille II, October, 1982.

[Col-87] A. Colmerauer, "Introduction to Prolog-III", ESPRIT'87, Achievments and Impact, Proc. of the 4th Annual ESPRIT Conference, Burussels, September 28-29, 1987, North-Holland.

[Din-87] M. Dincbas, H. Simonis, and P. van Hentenryck, "Extending Equation Solving and Constraint Handling in Logic Programming", ECRC Internal Report, IR-LP-2203, February, 1987.

[Hei-86] N. Heintze, J. Jaffar, C. S. Lim, S. Michaylov, P. Stuckey R. Yap, and C. N. Yee, "The CLP Programmer's Manual – Version 1.0", Department of Computer Science, Monash University, 1986.

[Hil-90] D. Hilbert, "Über die Theorie der algebraischen Formen", Math. Ann. 36, pp.473-534, 1890.

[JaL-86] J. Jaffar, and J-L. Lassez, "Constraint Logic Programming", IBM Thomas J. Watson Research Center, Internal Memo, 1986.

[JaM-85] J. Jaffar, and S. Michaylov, "Methodology and Implementation of a Constraint Logic Programming System", TR 54, Department of Computer Science, Monash University, June, 1985.

[SaA-88] 坂井 公、相場 亮、"CAL: 制約プログラミングの理論と実例"、情報処理学会ソフトウェア基礎論研究会、vol. 88、No. 8、88-SF-24、pp. 9–17、1988年2月12日.

[SaS-88] Y. Sato, and K. Sakai, "Boolean Gröbner Bases", LA シンポジウム,February, 1988.

[Tak-86] A. Takeuchi, and K. Furukawa, "Partial evaluation of Prolog Programs and Its Application to Meta Programming", Information processing 86, Dublin, North-Holland, pp. 415–420, 1986.

[Tam-84] H. Tamaki, and T. Sato, "Unfold/Fold transformation of Logic Programs", Second International Logic Programming Conference, Uppsala, 1984.