Vol. 30 No. 1　　情　報　処　理　　Jan. 1989

解　説

制約ロジック・プログラミング

——知識処理への新しいパラダイム——†

横井俊夫††　相場　亮†††

1. はじめに

近年，制約ロジック・プログラミング (constraint logic programming, rule-based constraint logic programming) が注目を集めている．これは，通常のロジック・プログラミングを知識処理に向けて拡張した新しいパラダイムである．しかしながら，制約 (constraint) という言葉のためか，とかく思い違いを生みやすく，この新しいパラダイムの意義が正しく理解されているとはいいがたい．以下にこのような誤解を生みやすい点を2点ほど指摘することにより，制約ロジック・プログラミングなるものの輪郭を描き出すことにする．

まず第一は，制約ロジック・プログラミングは，ロジック・プログラミングに制限を加えたものではないという点である．実際，意味するところはその逆である．制約ロジック・プログラミングは，通常のロジック・プログラミング（以下では，"通常の"という形容は省略する）を一部分（特殊形）として含む，より広い枠組みである．Prologの創始者であるColmerauerがProlog I[1]からProlog II[2]，そしてProlog III[3),4)]へと発展してきた経緯に典型的にみられるように，制約ロジック・プログラミングとはロジック・プログラミングの本来の姿を求めての帰着ともいうべきものである．

制約ロジック・プログラミングは，ロジック・プログラミングのままではその本質的な構造が捉えにくかった知識表現，意味表現，プログラム・仕様記述などの分野に新たな展開をもたらし，エキスパート・システム・シェル，自然言語処理，知的プログラミングなどの技術に強固な骨格を与えてくれるものである．

たとえば自然言語処理においては，単一化文法の拡張として，言語情報間の制約を導入する傾向が顕著であり，この分野と制約ロジック・プログラミングとの結び付きが注目されている．

強調したいことは，制約ロジック・プログラミングはプログラミング言語に対するパラダイムであるということである．したがってこの枠組みの中にはプログラム言語として多種多様のものが考えられ，実用化にごく近いところにまで仕上げられたものから，今後の研究課題となるものまで含まれる．

第二は，制約という考え方は人工知能にとって非常に広い役割を果たすことができるという点である．制約という言葉は，今までにもいろいろに使われてきた．人工知能の分野においても，ある特定の意味合いをもった用語として用いられてきた．たとえば，問題解決の一つの手法として，制約充足問題[38)]という問題の形式化と解法に関する歴史がある．また，最近ではエキスパート・システムについての議論の中で，システムを大きく設計型と診断型に分け，設計型には制約形式の知識が役立つうんぬんという説明がなされている．これらの既成概念が，本質の理解を妨げている．

ロジック・プログラミングにおいてはプログラム，知識，および問題の表現のための基本概念として"関係"を前面に打ち出した．この"関係"を構造化し，それに対する評価（計算・解釈・推論）の機構をも含めたものこそが"制約"という言葉の表そうとするものである．制約という考え方は，人工知能の中核概念として広く適用できる基本的な考え方である．これに加えて，今までのように"制約"だけが単独で扱われるのではなく，ロジック・プログラミングという柔軟で強力なプログラム言語に埋め込まれ，統合されていることにより，この一般性はさらに高められる．

いずれにせよ，制約ロジック・プログラミングは，ロジック・プログラミングのもつ利点と問題点の両方を受け継いでいる．

---

† Constraint Logic Programming——A New Paradigm for Knowledge Information Processing——by Toshio YOKOI (Japan Electronic Dictionary Research Institute, ltd.) and Akira AIBA (Institute for New Generation Computer Technology).

†† (株)日本電子化辞書研究所　本解説は，(財)新世代コンピュータ技術開発機構に在職中の成果に基づいてなされたものである．

††† (財)新世代コンピュータ技術開発機構研究所第一研究室

以下，2.においては，制約ロジック・プログラミングの定義を，その操作的モデルを用いた説明によって紹介する．3.においては，制約式とその評価機構を整理し，それに基づいて制約ロジック・プログラミング言語の国内・外の代表事例を紹介する．4.においては，これらの現状をふまえ，今後の課題について述べる．

2. 制約ロジック・プログラミングとは

ロジック・プログラミングは，1972年に Colmerauer によって提唱された Prolog をその端緒とするが，研究開発が本格化したのは，1982年にスタートした第5世代コンピュータ・プロジェクトからである．それ以降，Prolog を母体にさまざまな改良や拡張の努力が続けられている．それらの努力を方向付けるキーワードは“並列”と“制約”の二つであろう（もう一つ，対象指向の“対象”をあげることもできる）．

制約ロジック・プログラミングについても，やはり Colmerauer に負うところが大きい．Prolog II が1980年に発表され[2]，Prolog III は1984年にはかなりまとまった形となり，文献としては1987年に世に出された[3),4)]．これらの影響を受け，オーストラリア，アメリカ，ヨーロッパ，日本などで制約ロジック・プログラミングに関する研究開発が活発化してきている．

1988年4月，アメリカのロードアイランド州プロヴィデンス近郊で「言語と制約」と題されたワーク・ショップ (Workshop on Language and Constraints) が開かれた．このワーク・ショップには Colmerauer, Lassez, Jaffar ら，この分野でリーダーシップをとってきた研究者を始めとして，日本，アメリカ，フランスなどから 45 名の専門家が参加し，制約に関する最新の研究の発表があった．ここにおいても，新たな制約ロジック・プログラミング言語の提唱であるとか，その実装技術，あるいは並列制約ロジック・プログラミングなど，最近の研究の活発さを裏付ける発表が多数あった．

制約とは，歴史的には人工知能における，いわゆる「制約充足問題」という形で発展してきた．これは，制約を用いて，ある系における近接対象間の関係を記述し，それを弛緩法であるとか，伝播法であるとかの技法を用いて，その関係を満たすような対象の状態（たとえば変数の値）を求めるものである．すなわち，Colmerauer 以前の制約プログラミング言語は，これらの解決技法をどのように記述し，どのように制御するかという点に力点があったといえる．これらの制約充足問題やそのための言語については，文献[38)]の数章に良くまとめられた解説がある．

これに対し Colmerauer は，制約のもつ「宣言性」に着目した．これが Prolog の述語のある意味での拡張であることから，解決技法を利用者に意識させないで，すべての制約を宣言的に書くことのできる制約ロジック・プログラミングを提唱したのである．以下においては，制約といった場合，特に断わらないかぎり，この意味で用いる．このような立場からみると，ロジック・プログラミングのもつ非決定的計算とユニフィケーションは，制約自身，あるいはそのメタ制御との親和性が際立って高いということができる．

最も一般的に言って制約とは，ある領域における関係を記述したものである．そして，その関係の意味をさまざまなレベルで取り扱えるようにしたものである．ロジック・プログラミング言語においては関係を構文的に取り扱うのに対して，制約ロジック・プログラミングにおいては関係を意味レベルで扱うということができる．

このことを，以下にユニフィケーションを例として考えてみる．たとえば，“3+4”は構文的に考えると“3+4”，“X+4”，“X+Y”あるいは“3+Y”などとのユニフィケーションのみが可能である．一方，この式の算術式としての意味を考えると，たとえば“4+3”，“7”あるいは“12−X”などとのユニフィケーションも可能となる．このようなユニフィケーションはセマンティック・ユニフィケーション[18)]と呼ばれ，制約とその評価の一形態である．

二つの（一階の）項 t と s とのセマンティック・ユニフィケーションとは，ある与えられた計算領域 D における等式

$$t=_T s$$

の解を求めることであると，一般的に定義できる．ここで，T とは D において定義されているような等式理論である．容易に分かるように，計算領域をエルブラン空間とし，等式理論を空の理論とすれば，ロジック・プログラミングにおけるユニフィケーションを得ることができる．

このように，ある領域の意味的な関係式の記述とその取り扱いを可能にすることは，ロジック・プログラミングの記述力を飛躍的に向上させることになる．

2.1 スキーマ

制約ロジック・プログラミングの基本的メカニズム

を理解するために，制約ロジック・プログラミングのスキーマを単純化した操作的モデルを設定する．これは Dincbas[16]によるものである．

前述したように，制約ロジック・プログラミングはロジック・プログラミングを拡張したものである．したがって，以下に述べるスキーマもロジック・プログラミングのスキーマを拡張したものである．ロジック・プログラミングのホーン節，レゾルベント，評価に対応して，拡張ホーン節，拡張レゾルベント，拡張評価が設定される．

[拡張ホーン節]

拡張ホーン節を次のように定義する．

$P :- P_1, P_2, \cdots, P_n ; C_1, C_2, \ldots, C_k.$

ここで，$P_1, P_2, \cdots, P_n$ をリテラル部と呼び，$C_1, C_2, \cdots, C_k$ を制約部と呼ぶ．制約部がなければ通常のホーン節とまったく同じである．

[拡張レゾルベント]

次に，ホーン節の拡張に対応する拡張レゾルベントを定義する．拡張レゾルベントは通常のレゾルベントと，制約に対するレゾルベントの対となる．

$R = \langle RL : RC \rangle$

ここで，RL をリテラル・レゾルベント部と呼び，RC を制約レゾルベント部と呼ぶ．リテラル・レゾルベント部 RL は，ロジック・プログラミングのレゾルベントとまったく同一で，解かれるべきリテラルの論理積である．また，制約レゾルベント部 RC は解かれるべき制約をある種の標準形に直したものの論理積である．

[拡張評価]

この拡張ホーン節と拡張レゾルベントに対し，拡張評価を次のように定める．ある評価のステップにおける拡張レゾルベントが，次のようであったとする．

$R_n = \langle L_1, L_2, \cdots, L_m : RC \rangle$

拡張ホーン節

$P :- P_1, P_2, \cdots, P_k ; C_1, C_2, \cdots, C_l.$

に対し，$\theta(P) = \theta(L_1)$ となる代入 θ が存在するとする．また，

$RC' = \mathrm{simplify}(\theta(C_1 \& C_2 \& \cdots \& C_l \& RC))$

であるとする．ここで simplify は，true, false, undefined のいずれかと，簡約化された制約レゾルベントを値としてとるような関数で，制約の論理積の解(標準形)を求めるものである．このとき，$RC' \neq$ false ならば，新しいレゾルベント R_n+1 は次によって与えられ，評価が1ステップ先に進む．

$R_n+1 = \langle \theta(P_1, P_2, \cdots, P_k, L_2, \cdots, L_m) : RC' \rangle$

制約が互いに矛盾するならば，RC'＝false となり，この評価のパスは fail となる．このようにして評価を続け，リテラル・レゾルベント部が ϕ となったとき，評価は終了する．このときの制約レゾルベント部が解に付け加えられる．

2.2 例

本節では，簡単な例を使って上述のスキーマを説明する．ここでの例は，鶴と亀の頭の数と足の数との関係を記述したプログラムである．これを用いて鶴亀算を解くことを考える．プログラムを読みやすくするため，述語名，変数名はすべて日本語とする．また，変数はゴチック体で書くことにする．

鶴と亀(鶴, 亀, 頭, 足) :-**足の数(足, 鶴, 亀)**
; **頭＝鶴＋亀**. (1)

足の数(足, 鶴, 亀) :-
; **足＝4*亀＋2*鶴**. (2)

プログラムは二つの拡張ホーン節からなる．(1)の拡張ホーン節は，リテラル部も制約部も一つの式から，(2)は制約部のみからなっている．**亀**は亀の数を，**鶴**は鶴の数を，**頭**は頭の数を，そして**足**は足の数を表す．したがって，(1)の等式“**頭＝鶴＋亀**”は，頭の数は亀の数と鶴の数の和に等しいということを表す制約であり，(2)の等式“**足＝4*亀＋2*鶴**”は，足の数は亀の数の4倍と鶴の数の2倍の和に等しいということを表す制約である．

これに対して，たとえば次のようなゴールを実行することを考える．

?-鶴と亀(つる, かめ, 4, 10). (3)

すなわち，頭の数が4，足の数が10のときの亀と鶴の数を求めるゴールである．拡張レゾルベントの初期値は，次のとおりである．

$R_1 = \langle$**鶴と亀(つる, かめ, 4, 10)** : true$\rangle$ (4)

ここで，(4)のリテラル・レゾルベントと拡張ホーン節(1)との間の代入(ユニファイア) θ_1 は，次のようになる．

$\theta_1 =$ {**亀/かめ, 鶴/つる, 頭/4, 足/10**} (5)

また，制約レゾルベントは，次のようになる．

$\mathrm{simplify}(\theta_1((\text{頭} = \text{鶴} + \text{亀}) \,\&\, \text{true}))$
$= \mathrm{simplify}(\text{つる} + \text{かめ} = 4)$

ここで simplify を連立方程式を解く機能を表す関数とする．方程式1本では解けないので，そのままの式を返す．したがって，拡張レゾルベント R_2 は次のようになる．

R_2=〈足の数(10, つる, かめ) : つる+かめ=4〉 (6)

次に，(6)のリテラル・レゾルベントと拡張ホーン節(2)との間のユニファイア θ_2 は次のようになる．

θ_2={足/10, 亀/かめ, 鶴/つる} (7)

また，このとき制約レゾルベントは，次のようになる．

simplify(θ_2(足=4*亀+2*鶴 & つる+かめ=4))
=simplify((4*かめ+2*つる=10 &
つる+かめ=4))
=(かめ=1 & つる=3) (8)

したがって，拡張レゾルベント R_3 は次のようになり，評価が終わる．

R_3=〈ϕ, (かめ=1, つる=3)〉 (9)

このような評価を経た結果，亀の数1と鶴の数3が得られる．

3. 制約式と評価機構そして言語

制約ロジック・プログラミングというパラダイムのもとに，多種多様な言語が研究・開発されている．個個の言語を制約式とその評価機構によって整理し，国内外の事例を対応付ける．

3.1 制約式と評価機構

以下において，「制約式」とは制約を表現するための式あるいは言語を示し，「評価機構」とは制約の計算あるいは推論を行う機構を示す．この評価機構は前節のスキーマにおける関数 simplify に対応する．

評価機構は「遅延評価機構」と「簡約評価機構」の二つの部分からなる．簡約評価機構において用いられているアルゴリズムを「簡約アルゴリズム」と呼ぶ．

簡約とは，たとえば連立方程式における消去法や，二つの数値の同等性のチェックのように，制約式を利用者にとってより単純な形式へと変換する処理のことを言う．

記述可能な制約式でありながら簡約評価機構では処理できない場合，これをただちに簡約することはできず，具化（変数に具体的な値が代入されること）による制約式の簡単化によって，簡約可能な制約になるまで待つ必要がある．これが「遅延」である．たとえば簡約評価機構が制約式として線形の代数方程式を対象とする場合，非線形の代数方程式を制約式として許すと，変数の具化によって線形になるまで待たなければ簡約することは不可能である．このように，与えられた制約を簡約評価機構が扱えるようになるまで，その評価を行わせないのが，遅延機能の役割である．

もう一つ参考までに，制約を機能的にとらえた分類として，能動的制約と受動的制約という見方を紹介しておく[18]．直観的に言えば，受動的制約とは，どこかでだれかが候補となる値を生成してくれるのを待ち，それが制約に合うものかどうかを判定しようとするのに対し，能動的制約は，その制約式を満たす値をみずから求めようとするものである．この分類も，上の評価機構によって説明ができる．

制約式の評価における遅延と簡約の役割分担の，最も極端な場合を考えてみる．すなわち，制約の評価において簡約評価機構が分担する部分が非常に少なく，関係が成立するか否かの判定しか行わず，ほとんどの処理を遅延によって行うという場合である．これがこの分類における受動的制約である．一方，簡約評価機構が分担する部分がこれと比較して少しでも多い場合には能動的制約になる．

いずれにしろ，制約式にどのようなものを許すのか，遅延と簡約の境界をどこに置くか，そして簡約アルゴリズムに何を用いるかによって制約ロジック・プログラミング言語は特徴付けられ，分類される．

現在，利用可能な，あるいは利用を目的に研究されている代表的な制約式とその簡約アルゴリズムを整理すると，**図-1** のようになる．図-1 では，制約式をとりあえず6つのカテゴリに分類している．図の二重線は，制約式と簡約アルゴリズムの対応を示す．また，

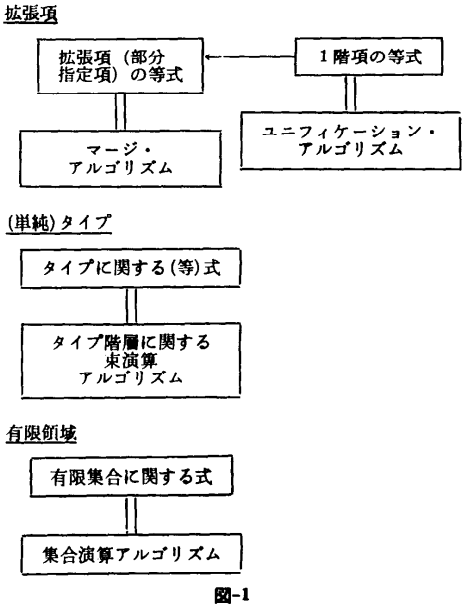

図-1

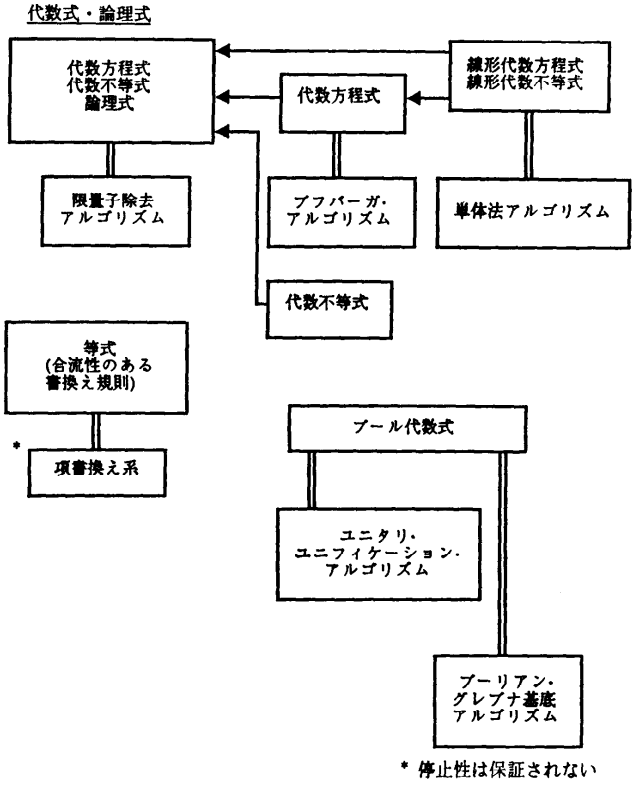

図-2

矢印は制約式間の相互関係を示し，左側にあるものほどより一般的な制約式であることを表す．

紙面の都合により，個々の制約式とアルゴリズムについての説明は割愛するが，あるアルゴリズムを簡約のために利用する場合に望ましい性質として，以下のものが考えられる．

（a） 簡約アルゴリズムは，完全性を失わずに，決定的に解けること．これは，結果として非決定的な計算はすべてロジック・プログラミングの部分で行われなければならないということを意味している．

（b） 簡約は，次々と標準形を計算していくような形で行われ得ること．簡約は計算ステップごとに少しずつ進められるからである．

（c） 簡約アルゴリズムの効率の良い実現法があること．

（d） 制約式はその言語の応用領域で十分頻繁に使われるものであること．

（e） 複数の制約式を組み合わせる場合には，それぞれの意味が明確に認識できること．

3.2 代表的な言語事例

制約式と評価機構の分類・整理を基に国内・外の代表事例を紹介する．ここにあげたものは，論文などを含め，入手し得る情報の多寡によって選び出したものである．

表-1 に8つの代表事例を，制約式にどのようなものを利用できるようにしているか，その評価に際し簡約評価機構として用いている簡約アルゴリズム，また遅延評価機構を用いている場合のその仕組，これら3要素で整理する．

以下に，代表事例のプロフィールを列挙する．（9）にそれ以外の事例を簡単に紹介する．

（1） CLP(ℜ)[11]

Jaffar, Lassez らによって提唱された言語スキーマ CLP(X)[10]に基づいて，1986年にオーストラリアの Monash 大学において設計・開発された言語である．CLP(X)は，この種の言語として多ソート一階論理に基づき，論理的枠組みの中で初めて意味論を与えたものであり，さらに satisfaction complete, solution compactness など，彼らの制約ロジック・プログラミング言語の枠組みに適合するための条件を明らかにした．

現在，Monash 大学からC言語で記述されたインタプリタの配布を受けることが可能である．C. Lassez, K. McAloon らによる，オプション・トレーディングに対する応用[9]を始め，多数の応用プログラムが書かれている．

（2） Prolog-II[2]

1980年に Colmerauer らによって，新しい Prolog のパッケージとして実装が始められたものである．この Prolog-II には，freeze を始めとする，制約ロジック・プログラミング言語における鍵となる技術が最初に取り入れられた．

処理系としては，1982年に Apple 上に実装されたものが最初で，その後，VAX や IBM-PC 上での実装も行われている．

（3） Prolog-III[3),4)]

Prolog-II の後に Colmerauer によって提唱，実現された言語である．

Prolog-III のアイデアは，Colmerauer が1984年に来日した際の ICOT における講演の中で発表され，

表-1 制約ロジック・プログラミング言語の代表例

言 語	制約式	簡約アルゴリズム	遅延評価機構
CLP(ℜ)	• 代数方程式 • 代数不等式	• 単体法	• 線形になるまで運転
Prolog-II	• 拡張項（有理木）間の等式 • freeze 述語により強化された Prolog 述語	• 拡張ユニフィケーション	• freeze 述語により陽に制御
Prolog-III	• 線形方程式 • 線形不等式 • ブール等式	• 単体法（?） • (?)	
CHIP	• 有限領域の等式 • ブール式の等式	• 有限集合演算 • ユニタリ・ユニフィケーション・アルゴリズム	
LOGIN	• 拡張項（Ψ項）の等式	• 拡張ユニフィケーション	
CIL	• 拡張項（部分指定項）間の等式 • freeze 述語により強化された Prolog 述語	• 拡張ユニフィケーション（マージ・アルゴリズム）	• freeze 述語により陽に制御
CS-Prolog	• 代数方程式 • 代数不等式	• 消去法 • sup-inf 法	• freeze 述語により陽に制御
CAL	• 代数方程式 • ブール等式	• ブフバーガ・アルゴリズム • ブーリアン・グレブナ基底アルゴリズム	

その時点で基本的な部分はかなりまとまっていたようである.

文献[4]によれば，Colmerauer らは，Prolog-III の処理系の市販を考えているようである．Colmerauer は非常に広い範囲への応用が可能であると述べているが，特に OR（オペレーションズ・リサーチ）や，エキスパート・システムをあげている.

（4） CHIP

これまで述べてきた各言語は，いずれも制約を陽に記述するものであったが，Dincbas[18] らの CHIP は，セマンティック・ユニフィケーションも用いており，制約は述語中に埋め込まれている.

処理系は，Dincbas らによって ECRC において実装されている．また，応用については，従前の制約充足問題，および論理回路の検証，設計，特殊化があげられており，CLP(ℜ) などと比較した場合，より古典的かつ大規模な制約問題を応用の中心に置いている.

（5） LOGIN[27),28)]

1985年に MCC の Aï-Kaci によって提唱された言語で，ロジック・プログラミングにタイプ階層を導入した言語である．すなわち，項を拡張し，インヘリタンス，すなわち is-a 関係をレゾリューションに基づいた推論機構にではなく，ユニフィケーションの中で扱う言語である.

応用については，知識ベースを指向している.

（6） CIL[22)]

ICOT の向井により，1985年に開発された言語で，状況意味論に基づいた自然言語処理のためのプログラミング言語であり，その特徴は，自然言語処理のためのデータ構造である部分指定項にある.

処理系に関しては，当初 DEC 2060 に実装されていたが，その後，1986年に逐次型推論マシン PSI に実装された.

（7） CS-Prolog[23)]

1987年に東京理科大学の溝口研究室において開発された言語で，データ駆動による遅延評価機構と，連立方程式を解くための消去法をユニフィケーションに組み込み，さらに領域概念の扱いを可能とすることによって，制約プログラミングに対処したものである.

実数領域をとる変数については，その大小関係に関する順序関係を基にして推論する方法を与えている.

制約充足問題を始めとしたエキスパート・システムについての応用を指向している.

（8） CAL[26)]

1987年に ICOT の坂井，相場らによって開発された言語で，任意次数の連立代数方程式，およびブール

代数等式を制約として記述し，評価することのできる言語である．

処理系は，当初 DEC 2060 上で開発されたが 1988 年には，逐次推論マシン PSI 上での実装が行われた．

図形定理の証明などを応用として考慮している．

（9） そ の 他

Constraint Prolog[24] は，1987 年に日本 IBM の沼尾によって開発された言語で，自由ブール代数を真偽値としてもつ Prolog, ProBoole[25] を基に作られた言語である．Constraint Prolog においては，ホーン節中に埋め込まれた制約パターンに対して，監視機構が付加されており，これがゴールの実行時にはたらいて，実行の制御をするようになっている．さらに制約の記述性を高めるために，遅延評価機構を組み込んでいる．Constraint Prolog は応用として，制約伝播機構に基づく問題解決を指向している．

ADL[20] は，建築，機械，電子回路などの設計に際しての設計支援を目的として作られたシステムを記述するための言語である．設計公式に代表される知識を一般に制約条件知識として捉え，リダクション手法により解探索を行う．ADL システムは，知識ベースエディタと制約リダクション系から構成される．設計知識ベースには，設計要求，設計結果，設計知識が記憶され，知識ベースエディタはこれらの編集・管理に利用される．一方制約リダクション系は設計計算，資料検索，設計の検証などに利用される．

TDProlog[21] は，項記述と呼ばれる一種の制約を導入した言語である．項記述というのは，項に対してその成立条件としての制約の記述のことである．たとえば，ある変数の成立条件として，ある述語を記述する．項記述は変数以外の項とユニファイされた時点で起動されるため，述語の遅延評価が可能となる．

CRL[39] は，1987 年に ICOT の横田によって提唱された，演えきデータ・ベースの実現のための言語である．その特徴は，データ構造として集合を許している点にある．処理系については，現在検討中である．

これら以外にも，E-ユニフィケーションによって，ロジック・プログラミング言語と関数型言語の融合をはかった制約ロジック・プログラミングがある．たとえば，Uniform[29], Equality for Prolog[30], Kernel Leaf[31], Lassez らの言語[32], Van Emden らの言語[33], SLOG[34], NARROWER[35], Talos[36], 山本の言語[37] などがある．E-ユニフィケーションとは，ある書換え規則によって定まる等号論理上で等価なものを同一視するような，セマンティック・ユニフィケーションの一種である．

4. 今後の課題

現状の制約ロジック・プログラミング言語のもつ問題点として，

- 大規模なアプリケーションからのフィードバックがない，
- 制約式の評価機構の実行速度の遅さが，処理系を効率の悪いものにしている．
- 制約とその適用領域について未整理である，

などが指摘されている．

ここでは，制約ロジック・プログラミングに対する批判への解答，また現状の人工知能研究へのコメントなどを込めて，より広い観点から今後の課題を整理する．

（1） ロジック・プログラミングと共有する課題

制約ロジック・プログラミング言語は，前にも述べたように，ロジック・プログラミングの拡張であり，したがってロジック・プログラミングが内包する問題点をそのまま引き継いでいる．たとえば，大規模なソフトウェア作成に不可欠なモジュール化機能の貧弱さがある．これらに関しては，ロジック・プログラミングに施されたさまざまの拡張をほぼそのまま導入することができると考えられる．ただし，より適合したものを選ぶ努力を怠ってはならない．対象指向プログラミングとのすり合わせなどもこの路線に沿って進めることができるであろう．

（2） ロジック・プログラミングとの分担

制約ロジック・プログラミング言語をも含めた全体の枠組みの中で，ロジック・プログラミングをどう位置付け，育ててゆくかということも重要な課題である．ロジック・プログラミング言語は，質問の関係という，最も単純な制約を対象とするところから，制約ロジック・プログラミング系言語にとってのアセンブラ言語，あるいはインプリメンテーション言語と位置付けられる．したがって，制約ロジック・プログラミング言語のためのアーキテクチャを考える立場からは，ロジック・プログラミングのためのアーキテクチャ＋α に焦点を合わせれば良いことになる．α は，各種の割出しの機能などで，制約評価のより効率の良い実現を可能にする仕組みである．なお，並列ロジック・プログラミングを土台としたとき，これらの事情がどうなるかは，興味深い今後の課題である．

（3） 高速化へ向けて

大規模なプログラムの記述が試みられるようになるにつれ，処理速度の一層の向上が急務の課題となってきた．対策としては大きく三つが考えられる．制約式の評価アルゴリズムの改良，並列評価アルゴリズムの開発による高速並列処理マシンの活用，プログラム変換・最適化技法によるプログラムの最適化・高速化である．三番目の対策は最も興味深いもので，次にも取り上げる．

（4） メタ評価・部分評価技術の確立に向けて

複雑かつ一般的な制約式を評価するとすれば，評価機構は複雑となり，遅くなるのは当然である．そのように割り切る観点から考えると，このような制約式をもつ制約ロジック・プログラミングを一種の仕様記述としてみなすことを考える．すなわち，実行可能な仕様とみなすわけである．そうすれば，少々遅くとも十分に役割を果たしてくれているのではなかろうか．どのような種類の制約式を，どのように混在させるか，さらにこれを特殊なものから一般的なものへと系統立てて考えることにより，制約ロジック・プログラミングは仕様的なものからプログラム的なものまでの，一連の言語の系列と考えることができる．これらが同族の言語であるということにより，仕様記述とその実現に関する技術に新しい可能性を開くことができると考える．

そして，仕様からプログラムへの変換の過程を司るものが，プログラム変換，特にメタ評価・部分評価の技術である．メタ評価・部分評価の技術は，基礎的なところは整理されつつあるが，もう一つの飛躍が望まれている．制約ロジック・プログラミング系の言語群のように統一的な枠組みで捉えることのできる言語系列に対して適用することにより，これらの技術を改良し，拡張していくことが大きな鍵となる．

（5） エキスパート・システム・シェル技術の確立に向けて

エキスパート・システム作りとは，高級化されたプログラム言語によるプログラム作りであると考える．この観点に立てば，安定した標準的なプログラム言語の開発と，高度で知的な支援をしてくれるプログラミング・システムの開発が要となる．

現在，世の中に商品化され，使われているエキスパート・システム・シェル，あるいはAIツールと称されるものは，とうていその任に耐えうるものではない．これへの反動として，実用化のためにFORTRANやC言語を用いるという議論は，機械語の時代に逆戻りしようというに等しい．このような傾向がエキスパート・システムの開発が低迷している要因の一つとも思われる．

安定した標準的なプログラム言語，その一番の有力な候補は制約ロジック・プログラミングにある．そして，メタ評価・部分評価による知的プログラミング技術が作成支援システムの核となるはずである．

5. ま と め

人工知能研究の中心的課題の一つは，問題や知識の表現とその評価（計算・解決・推論）機構である．そして人工知能の他の重要な課題と同様，この課題も，一般化と特殊化との間を行きつ戻りつしながら研究が行われてきた．

表現可能な対象が非常に一般的で，かつそれに対する評価機構がきわめて強力かつ効率も良いというようなものはありえない．

（1） 評価機構の機能を追求してより抽象度の高い表現を評価できるようにしようとすれば，表現可能な対象をその抽象度の高さゆえにごく限らなければならない．また，

（2） 表現の一般性を追求してどのようなものでも書けるようにすれば，評価機構自体はごく単純なものとしなければならない．

(1)は，各種の論理系や証明アルゴリズムを代表例とするものにまとめられ，述語論理，様相論理，ファジィ論理などの多数の論理系，ならびに各種の数式とそれらに対する処理アルゴリズムとなった．すなわち，高度な対象を表現しようとする分，処理可能な表現はそれぞれの論理式や数式に限られている．

一方，(2)はプログラミング言語にまとめられ，Lisp や Prolog に代表される人工知能向言語となった．すなわち，プログラミング言語の表現可能な対象は非常に一般的なものであるがゆえに，その表現は比較的単純なものの組合せによって記述され，論理式や数式の表現に際しては，コード化が必要となる．

次に進むべきは，この両者をいかにバランス良く融合し，互いに補完しあえるようにするかという方法を探ることである．知識表現という表題のもとで，ここ十数年このような模索が行われてきた．結果は，もう一つ決め手に欠けるというものであった．

これを解決する決め手が，制約ロジック・プログラミングであると考えている．抽象度の高い，複雑な対

象を制約という形で記述し，一般性についてはこれをロジック・プログラミングの部分に依存しており，これが意味論によって結び付けられ，統合化されているというのが制約ロジック・プログラミングなのである．これが本解説のサブタイトルが意味するところである．

謝辞 本稿の作成に際し，ICOTの長谷川隆三氏，向井国昭氏，坂井公氏，大木優氏（現日立製作所），坂根清和氏，横田一正氏，大須賀昭彦氏を含め多くの方に謝意を表する．

参考文献

以下に，制約ロジック・プログラミングに関する参考文献をあげる．ただし，制約ロジック・プログラミング自体，非常に新しい研究分野であるため，公式に出版された文献がまだそれほど多くない．中には，内部レポートのように若干入手しにくいものも含まれる．また，CLP(ℜ)については，多くの文献があげてあるが，内容的に重複するものも多い．

なお，制約ロジック・プログラミングについての論文は，FGCS '88, ICLP, SLP (1989年は NACLP—North American Conference on Logic Programming という名称で開催されるそうである) を始め，ロジック・プログラミング関係の学会で発表されることが多い．

[Prolog]

1) Robinson, J. A.: Logic Programming—Past, Present and Future—, New Generation Computing, Vol. 1, No. 2, pp. 197-124 (1983).

[Prolog-II]

2) Colmerauer, A.: Equations and Inequations on Finite and Infinite Trees, Proc. of FGCS '84 (1984).

[Prolog-III]

3) Colmerauer, A.: Introduction to Prolog III, ESPRIT '87 Achievements and Impact, Proc. of the 4th Annual ESPRIT Conference, Brussels, September 28-29, North-Holland (1987).

4) Colmerauer, A.: Opening the Prolog III Universe, BYTE, August 1987, pp. 177-182 (1987).

[Constraints]

5) Sussman, G. J. and Steele, G. L. Jr.: CONSTRAINTS: A language for Expressing Almost-Hierarchical Descriptions, Artif. Intell., Vol. 14, pp. 1-39 (1980).

[CLP]

6) Heintze, N., Jaffar, J., Lim, C., Michaylov, S. Stuckey, P., Yap, R. and Yee, C.: The CLP Programmer's Manual Version 1.0, Internal Memo, Dept. of Computer Science, Monash Univ. (June. 1986).

7) Heintze, N., Jaffar, J., Lassez, C., McAloon, K., Michaylov, S., Stuckey, P. and Yap, R.: Constraint Logic Programming—A Reader, 4th IEEE Symposium on Logic Programming, San Francisco, 31-Sep. 4 (Aug. 1987).

8) Heintze, N., Michaylov, S. and Stuckey, P.: CLP(ℜ) and Some Electrical Engineering Problems, Proc. of the 4th International Conference on Logic Programming, pp. 675-703 (1987).

9) Huynh, T. and Lassez, C.: A CLP(ℜ) Option Analysis System, Internal Memo, IBM Thomas J. Watson Research Center.

10) Jaffar, J. and Lassez, J. L.: Constraint Logic Programming, Internal Memo, IBM Thomas J. Watson Research Center (Aug. 1986).

11) Jaffar, J. and Michaylov, S.: Methodology and Implementation of a CLP System, Proc. of the 4th International Conference on Logic Programming, pp. 196-218 (1987).

12) Jaffar, J. and Lassez, J. L.: From Unification to Constraints, to appear in LNCS.

13) Lassez, C.: Constraint Logic Programming, BYTE, pp. 171-176, (Aug. 1987).

14) Lassez, C.: Constraint Logic Programming, Internal Memo, RC 12589, (log #56656) 3/17/87, IBM Thomas J. Watson Research Center (Mar. 1987).

15) Stuckey, P.: On the Foundation of Constraint Logic Programming, Internal Memo, Dept. of Computer Science, Monash Univ. (Aug. 1987).

[CHIP]

16) Dincbas, M.: Constraints, Logic Programming and Deductive Databases, Proc. of France-Japan AI Symposium '86, pp. 1-27 (1986).

17) Van Hentenryck, P. and Dincbas, M.: Domains in Logic Programming, Proc. of AAAI, pp. 759-765 (1986).

18) Dincbas, M., Simonis, H. and Van Hentenryck P.: Extending Equation Solving and Constraint Handling in Logic Programming, ECRC Internal Report IR-LP-2203, (Feb. 1987).

19) Van Hentenryck, P. and M. Dincbas: Forward Checking in Logic Programming, Proc. of the 4th International Conference on Logic Programming, pp. 229-256 (1987).

[ADL]

20) 長澤 勲，古川由美子，荒巻重登：論理プログラミングを基礎とした設計システム記述言語 ADL，情報処理，Vol. 25, No. 4, July, pp. 606-613 (1984).

[TDProlog]

21) Nakashima, H.: Term Description—A Simple Powerful Extrension to Prolog Data Structures, Proc. of IJCAI (1985).

[CIL]
22) Mukai, K.: A System of Logic Programming for Linguistic Analysis, to appear in ICOT-TR.

[CS-Prolog]
23) 川村十志夫, 大和田勇人, 溝口文雄: CS-Prolog —拡張単一化に基づく CONSTRAINT SOLVING, Proc. of the Logic Programming Conference '87, pp. 21-28 (1987).

[Constraint Prolog]
24) 沼尾雅之: Update Propagation Network—制約プログラムのための枠組, Proc. of the Logic Programming Conference '87, pp. 29-36 (1987).

[ProBoole]
25) Morishita, S., Numao, N. and Hirose, S.: Symbolical Construction of Truth Value Domain for Logic Program, Proc. of 4th ICLP, pp. 533-555 (1987).

[CAL]
26) Sakai, K. and Aiba, A.: CAL—Theoretical Background of Constraint Logic Programming and Its Applications, to appear in ICOT TR.

[LOGIN]
27) Aï-Kaci, H.: Logic and Inheritance, MCC TR AI-076-85 (1985).
28) Aï-Kaci, H.: LOGIN—A Logic Programming Language with Built-in Inheritance, J. Logic Program., pp. 185-215 (1986).

[Logic Programming with Equation]
29) Kahn, K. M.: UNIFORM—A Language Based upon Unification which Unifies (much of) Lisp, Prolog, and Act 1, Proc. of IJCAI-81, pp. 933-939 (1981).
30) Kornfeld, W. A.: Equality for Prolog, Proc. of IJCAI-83, pp. 514-519 (1983).
31) Giovannetti, E., Levi, G., Moiso, C. and Palamidessi, C.: Kernal Leaf—An Experimental Logic Plus Functional Language—Its Syntax, Semantics and Computational Model, ESPRIT Report.
32) Jaffar, J., Lassez, J. L. and Maher, M.: A Theory of Complete Logic Programs with Equality, Proc. of FGCS 84, pp. 175-184 (1984).
33) Van Emden, M. H. and Yukawa, K.: Logic. Programming with Equations, Dept. of Comp. Sci., Univ. of Waterloo, TR CS-86-05 (Oct. 1986).
34) Fribourg, L.: SLOG: A Logic Programming Language in Interpreter Based on Clausal Superposition and Rewriting, Proc. of SLP, pp. 172-184 (1985).
35) Rety, P., Kirchner, C., Kirchner, H. and Lescanne, P.: NARROWER—A New Algorithm for Unification and Its Application to Logic Programming—, Proc. of Rewriting Techniques and Applications (LNCS 202), pp. 141-157 (1985).

[TALOS]
36) Kanamori, T.: Features of a Meta-Unification Based Language Talos, ICOT TR-151 (1984).

[Narrowing]
37) Yamamoto, A.: A Theoretical Combination of SLD-Resolution and Narrowing, Proc. of 4th ICLP, pp. 470-487 (1987).

[制約充足問題]
38) Leler, W.: Constraint Programming Language—Their Specification and Generation, Addison-Wesley Publishing Co., ISBN 0-201-06243-7 (1988).

[CRL]
39) Yokota, K.: Deductive Approach for Nested Relations, to appear in ICOT TR (1988).

(昭和63年9月12日受付)