人 工 知 能 78－7
（1991. 9. 12）

# 制約論理プログラミング言語Keyed CLPとそのOR/MSにおける意思決定問題への応用

平石 邦彦
（株）富士通研究所 国際情報社会科学研究所

本研究では，OR/MSなどで扱われる数理計画問題，あるいは，管理会計や経理業務などで扱われる意思決定問題に対し，制約論理プログラミング言語の枠組みを利用した解法を提案する．制約論理プログラミング言語の処理系として，各述語に固有のキー(Key)を付けたKeyed CLPを作成し，実際の問題を解かせることにより，その有効性を確認した．キーは各述語を１つの関係として見たときのキー属性に対応する．この拡張により，計算効率を改善することができる．Keyed CLPはまた，線形の目的関数を最適化するための組み込み述語をもっており，これにより，線形計画問題を容易に解くことができる．

# A Constraint Logic Programming Language Keyed CLP and Its Applications to Decision Making Problems in OR/MS

Kunihiko Hiraishi

International Institute for Advanced Study of Social Information Science
FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan

The aim of this research is to utilize constraint logic programming (CLP) for solving decision making problems in Operations Research / Management Science. In this research, a new constraint logic programming language Keyed CLP is developed. Keyed CLP has some special features for solving problems in OR/MS. Key arguments are attached to each predicate. Each key represents the functional dependency in the relation, and is used for improving computational efficiency. In addition, Keyed CLP has built-in predicates for solving linear optimization problems.

# 1. はじめに

本研究では，OR(Operations Research) / MS(Management Science)などで扱われる数理計画問題，あるいは，管理会計や経理業務などで扱われる意思決定問題に対し，制約論理プログラミング言語の枠組みを利用した解法を提案する．制約論理プログラミング言語の処理系はCLP(ℛ) [Jaffer 87]，CHIP [Dincbas 88], CAL [Aiba 88]など，すでにいくつか発表されているが，本研究では，制約論理プログラミング言語の処理系としてKeyed CLPを開発し，実際の問題を解かせることにより，その有効性を確認した．Keyed CLPは，各述語に固有のキー(Key)を付けることを特徴としている．キーは各述語を１つの関係として見たときのキー属性に対応する．この拡張により，計算効率をかなり改善することができる．

OR／MSなどで扱われる問題の特徴として，以下のことが挙げられる．

(i) 各対象間の数値的な制約式の集まりにより問題が記述される．

(ii) 目的(Objectives)が設定される．制約を満たす入力パラメータの中で目的関数を最大（最小）にするものを求めよ，といった形式の最適化問題が多い．また，複数の目的関数をもつ場合もある．

(iii) 数値制約は線形のものが多い．

(iv) 多くのパラメータをもち，それらを同時に決定することは困難である．したがって，試行錯誤的な問題解決が必要になる．

このような問題に対し，記号的制約／数値的制約の宣言的記述が可能で，かつ，その解法も処理系がそなえている，すなわち，ユーザーが解法の手順をプログラムとして記述する必要のない，制約論理プログラミングの枠組みは有効であると考えられる．また，(ii)の要求に対しKeyed CLPでは，線形の最適化問題を解くための機能をもっている．

OR/MSの分野では，既存のモデルを計算機上に蓄積し，必要に応じて検索し利用するというモデルベース管理システムの考え方もいくつか提案されてきた[Dolk 86]．また，問題の抽象的レベルの構造の記述，それを具体化するためのパラメータ，問題の解法，さらに，問題の記述と解法を結ぶインタフェースをそれぞれ分離して記述しようという考え方が提案されている[Geoffrion 87]．制約論理プログラミング言語の枠組みは，これらの機能をワークステーション程度の小規模なシステムで実現するための，有効な手段の１つであると考えられる．

以下，第２章ではKeyed CLPの説明を行い，第３章ではKeyed CLPを用いた問題解決の例について述べる．最後に第４章では，Keyed CLPのもつ特徴についての考察を行う．

# 2. Keyed CLPの概要

## 2.1 キー付き述語の導入

Keyed CLPは，UNIX上でC言語により記述された制約論理プログラミング言語である．Keyed CLPの特徴として以下の形をしたキー付き述語の存在がある．

述語名($Key_1$, ... , $Key_n$ : $Arg_1$, ... , $Arg_n$)

述語を１つの関係(relation)として見れば，キー付き述語はキー属性を $Key_1$, ... , $Key_n$ とする関係のタプルとして考えることができる．１つの関係内において，キー属性の値を指定すれば関係のタプルは一意に決定される．Keyed CLPの実行においてもこの性質が保存される．すなわち，キー付き述語のキーは，関係がもつ関数従属性を陽に表現したものである．簡単な例によりキー付き述語の働きを見てみよう．

```
p(K : A).
q(K, X, X + Y) :- p(K : X), p(K : Y).

| ?- q(key, 3, Z).
Z = 6

*** yes ***
```

Keyed CLPでは，１つのゴールの実行において，同一のキー属性値をもつ述語の引き数はすべて同じ値をとる．したがって，p(key : Y)が呼ばれるときには，すでにキーの値keyをもつ述語p(key : 3)が決定しているので，Yには3が束縛される．この例では，キー付き述語p(K : A)（Kがキー属性）は定数３を格納する大域変数として働いており，Prologがもつ組み込み述語assertを用いたプログラム q(K, X, X + Y):- assert (p(K, X)), p(K, Y) と同じ働きをする．

キー付き述語には定数だけでなく，制約条件付きの変数も保存される．つぎの例を見てみよう．

```
a(1).
a(2).
p(K : A, B):- A >= B.
q(K, Z, A + A1):-  p(K : A,  Z), a(A1), p(K : A1,  _).

| ?-  q(key, 2, B).
B = 4

*** yes ***
```

p(key : A, 2) と規則 p(K : A, B):- A >= B の単一化に

より，キー付き述語 p(key : A, 2):- A >= 2 が得られる．これと p(key : 1, _) の単一化は，制約1≥2 が成り立たないので失敗する．したがって，解はA = A1 = 2のときのB = 4 だけとなる．このような機能はassertだけでは実現することはできない．

キー付き述語の導入には，会計，経理などで扱われる問題を効率よく実行する，という目的がある．会計，経理なので扱われる問題では，計算により求められた1つの値が，別の値を計算するために何度も用いられる，といったことがしばしば起こる．たとえば，予算計画をたてるときには，製品の原価，販売数などが基本データとして何度も参照されることになるが，これらをsales_units(a, X) :- ... のようにキーのない述語で記述すると，この述語が呼び出されるごとにbody部が評価されてしまう．しかし，販売数の値は製品名が与えられれば一意に決まるものであり，一度計算されれば，その値を参照するだけでよいはずである．キー付き述語を用いてsales_units(a : X) :- ...と記述することによりこの問題を避けることができる．

さらに，プログラムを記述する上では，キー付き述語を大域変数の格納場所として用いることにより，述語の引き数の数を減らすことができる．キー付き述語を用いないと，変数はすべて局所的なので，変数から変数へ値を受け渡すためには，述語の引き数としてその変数を持たせなくてはならない．

フィボナッチ数を計算するプログラムを例にして，キー付き述語により，どのように計算が効率化されるかを見てみよう．

```
fib(0 : 1).
fib(1 : 1).
fib(N : X1 + X2) :-
    N > 1, fib(N - 1 : X1), fib(N - 2 : X2).
```

ゴール fib(5 : X) は図1に示す順序で実行される．各ノードに付けられた数字は，その述語が実行される順序を表している(depth-firstなのでこのようになる)．

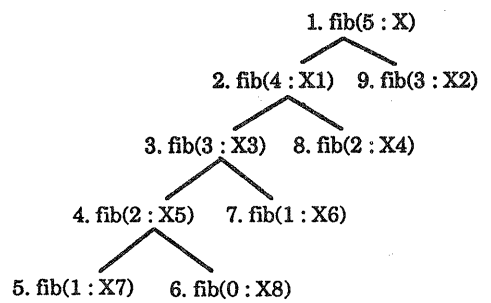


図1 fib(5 : X)の実行

fib(5 : X)はfib(4 : X1)とfib(3 : X2)により決定される．また，fib(4 : X1)はfib(3 : X1)とfib(2 : X2)により決定される．ノード7，8，9の各述語は，すでに同じキーをもつ述語がそれぞれノード5，4，3で得られているので，body部の評価は行なわれない．

## 2.2 Keyed CLPの操作モデル

Keyed CLPでは数値的な制約のことを特に制約とよぶ．以下に示す制約を素制約とよぶことにする．

Expr = Expr, Expr ≥ Expr, Expr ≤ Expr,
Expr > Expr, Expr < Expr

ここで，Exprは変数，定数，+, -, *, /, (, ) からなる．変数の定義域は実数領域である．Keyed CLP プログラムは有限個の規則から構成される．各規則はつぎの形をしている．

$$A_0 :- c_1, c_2, \ldots, c_n, A_1, A_2, \ldots, A_m.$$

ここで，$c_i\,(i = 1,\ldots,n)$ は素制約，$A_j\,(j = 1,\ldots,m)$ は項である．Keyed CLP の実行のある時点におけるゴールはつぎの形をとる．

$$(c_1, c_2, \ldots, c_q\ ; A_1, A_2, \ldots, A_r\ ; \theta\ ; S)$$

ここで，$c_i\,(i = 1, \ldots, q)$ は素制約，$A_j\,(j = 1, \ldots, r)$ は項，θは束縛，$S$は既に決定されたキー付き述語の集合である．初期状態では$S$は空である．いま，$G_1$をつぎのような空でない素制約の集合$C_1$を含むゴールとする．

$$G_1 = (C_1\ ; A_1, A_2, \ldots, A_m; \theta\ ;\ S)$$

$C_1$が可解(solvable)なとき，つぎの形のゴール$G_2$と置く．

(i) $A_1$がキー付き述語以外の場合
プログラムPがつぎの形の規則を含むとする．

$$B_0 :- C_2, D_1, D_2, \ldots, D_m.$$

ここで，$A_1\theta$と$B_0$は単一化可能とする．

$$G_2 = (C_1, C_2, [A_1\theta = B_0]\ ; B_1, B_2, \ldots, B_m, A_2, \ldots, A_m;\ \theta \cup \{A_1\theta = B_0\}\ ;\ S)$$

新しい制約は$C_1$,$C_2$ および$A_1\theta$と$B_0$の単一化によって

生じる制約 $[A_1\theta = B_0]$の和，新しい束縛は$A_1\theta$と$B_0$の単一化によって生じる束縛 $\{A_1\theta = B_0\}$を$\theta$に加えたものになる．$C_1$, $C_2$, $[A_1\theta = B_0]$ は可解でなければならない．

(ii) $A_1$がキー付き述語の場合

$A_1\theta$は変数を含んでいてはならない．そして，

(ii - a) $S$に$A_1\theta$と同じ述語名，同じキーの値をもつ述語が含まれていない場合：

プログラム$P$がつぎの形の規則を含むとする．

$$B_0 :- C_2, D_1, D_2, \ldots, D_m.$$

ここで，$A_1\theta$と$B_0$は単一化可能とする．

$$G_2 = (C_1, C_2, [A_1\theta = B_0]\ ; B_1, B_2, \ldots, B_m, A_2, \ldots, A_m;\ \theta \cup \{A_1\theta = B_0\}\ ;\ S \cup \{A_1\theta\})$$

新しい制約は$C_1$,$C_2$ および$A_1\theta$と$B_0$の単一化によって生じる制約 $[A_1\theta = B_0]$の和，新しい束縛は$A_1\theta$と$B_0$の単一化によって生じる束縛 $\{A_1\theta = B_0\}$を$\theta$に加えたものになる．また，$A_1\theta$が$S$に加えられる．

(ii - b) $S$に$A_1\theta$と同じ述語名，同じキーの値をもつ述語$D$が含まれている場合：

$$G_2 = (C_1, [A_1\theta = D]\ ; A_2, \ldots, A_m\ ;\ \theta \cup \{A_1\theta = D\}\ ;\ S)$$

ここで，$A_1\theta$と$D$は単一化可能であり，さらに，$C_1$, $[A_1\theta = D]$は可解であるとする．もしそうでないときは，サブゴール$A_1$は失敗する．

このような$G_2$が存在するとき$G_2$は$G_1$から導出可能であるという．導出列とはゴールの有限あるいは無限の系列で，各ゴールが前のゴールから導出可能であるものをいう．成功列とは有限の導出列で，最後のゴールが制約およびキー付き述語の集合だけからなるものをいう．１つの導出列中に現われる，同一のキー属性値をもつ述語の引数の値はすべて同じ値をもつことになる．

## 2.3 解法

各導出段階では，既存の制約と新たに追加される制約が矛盾しないかどうかを調べることになるが，Keyed CLPではCLP($\mathcal{R}$)等と同様に，線形計画法で用いられるシンプレックス法により制約が可解であるかどうかを調べている．非線形制約は評価が遅延され，変数への値の代入などにより線形化された時点で他の制約との整合性が調べられる．最後まで線形化されずに残った制約は，数式の形で出力される．

Keyed CLPは，基本的には線形の制約しか解かない．しかし，Keyed CLPが対象とする問題の領域がOR/MSで扱われる問題であることから，線形制約を解ければかなりの問題領域をカバーできると考えられる．また，現実の問題をモデル化するときには不等式制約は必須であると考えられる．なぜなら，値が厳密な形で指定されることは稀で，許容される領域の形で表現されることが多いからである．非線形制約を扱おうとすると，不等式制約の導入が困難になる．

## 2.4 高階述語

Keyed CLPではシンプレックス法により，線形の目的関数に対する最適解を求めることができる．これは，つぎの述語により行なわれる．

```
min(Expr, Goal)
max(Expr, Goal)
```

Exprは線形の式である．述語min(max)は，Goalを満足し，かつ，Exprを最小（最大）にする変数の束縛を求める．これらの述語は以下の式と等価である．

$$\mathrm{Expr} = v,\ \mathrm{Goal}$$

ここで，$v$はGoalに示される制約条件の下でのExprの最大（最小）値である．Goalに複数の解が存在する場合は，可能なすべての場合について最適化を行ない，さらにその中で最適なものを最終的な解とする．これにより，制約が表現する実数空間上での領域が凸領域の有限個の和集合となっている場合についても最適解を求めることができる．例としてつぎのプログラムを考える．

```
region(X, Y):- X >= 0, Y >= 0, X + Y <= 1.
region(X, Y):- X >= 0, X <= 2, Y >= 0, Y <= X.

|?- max(Y, region(X, Y)).

X = 2
Y = 2

*** yes ***
```

region(X, Y)で示される領域は，２つの凸領域の和集合になる（図２）．述語maxは，それぞれの領域に対して変数Yの最大値を求め（最初の領域ではY = 2，２番目の領域ではY = 1が最大値），それらの中で最大

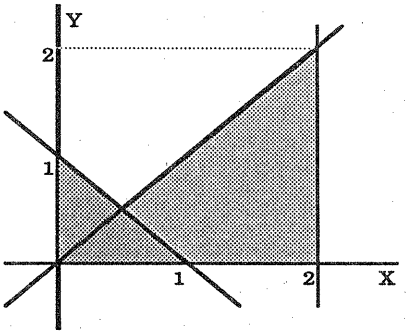


図2 region(X, Y)が表す領域

のものを解としている．複数解を探索するときには，既に求められた解を探索木の枝刈りに用いる分岐限定操作を行っている．また次章の例で述べるが，min, maxはネストさせることができる．これにより，min-max (max-min) 型の最適化問題を解くことができる．その他，つぎの高階の述語が用意されている

sum(Expr, Goal, Res) : Goalを成功させるすべてのExprの値の合計をResと単一化する．
avg(Expr, Goal, Res) : Goalを成功させるすべてのExprの値の平均をResと単一化する．

## 3. Keyed CLPによる問題解決の例

### 3.1 生産計画モデル

付録1のプログラムは典型的な線形計画問題のモデル化の例である．a, b, c, dの4種類の製品があり，それぞれ生産に必要な資源（原材料，電力，人員）が与えられている．これらは製品名をキーとするキー付き述語により表現されている．また，各資源には，使用可能量の上限が与えられている．資源の制約を満足し，かつ，利益を最大化するような各製品の生産量を求めることを目的とする．さらに，述語constraintでは，つぎのような制約が追加されている．

(Pb = 0; Pc = 0; Pd = 0)

ここで，変数Pb, Pc, Pdは，それぞれ製品b, c, dの生産量を表している．この制約は，製品a, b, cのうちの少なくとも1種は生産しないことを意味している．この制約条件により，実行可能解の領域は3つの凸領域（それぞれPb = 0, Pc = 0, Pd = 0に対応する）の和集合となる．述語ProductはTotal_ProfitおよびResource_Constrでも参照されているが，生産量が非負であるという条件X >= 0は最初に呼び出されたときしか評価されない．述語Feasibleは実行可能解を表わす．

```
| ?- Feasible(S, Pa, Pb, Pc, Pd).
S = 555.000000 -3.000000 * _20 + 11.000000 *
_143 + 9.000000 * _142
Pa = 25.000000 + _142
Pb = 0.000000
Pc = _20
Pd = 30.000000 - _20 + _143
_20 >= 0
30.000000 - _20 + _143 >= 0
227.500000 -4.000000 * _20 -2.500000 * _142 -
2.000000 * _143 >= 0
235.000000 -4.000000 * _20 -5.000000 * _142 -
3.000000 * _143 >= 0
15.000000 + 3.000000 * _20 -3.000000 * _142 -
5.000000 * _143 >= 0
_142 >= 0
_143 >= 0
_142 >= -25.000000

*** yes ***

Retry? y

S = 555.000000 -6.000000 * _18 + 11.000000 *
_143 + 9.000000 * _142
Pa = 25.000000 + _142
Pb = _18
Pc = 0.000000
Pd = 30.000000 - _18 + _143
...
```

バックトラックにより，それぞれの領域に対応する実行可能解を順次求めることができる．さらに，最適解は述語maxにより求められる．

```
| ?- max(S, Feasible(S, Pa, Pb, Pc, Pd)).
S = 740.000000
Pa = 53.333333
Pb = 0.000000
Pc = 23.333333
Pd = 6.666667

*** yes ***
```

### 3.2 PERT線図によるスケジューリング

PERT線図を用いたスケジューリング問題について考える．図3に示すPERT線図は自動車の設計から生産に至るまでの工程を表したものである[Willis 87]．各アーク上の数字は，その工程に必要な期間を表している．PERT線図の解析でまず行なわれるのは，最早開始時刻(Earliest Start Time)および最遅終了時刻(Latest Finish Time)を求め，クリティカルな工程（その工程が少しでも遅れると全体の遅れにつながるような工程）を見つけることである．最早開始時刻および最遅終了時刻はつぎのようにmin，maxを用いて簡潔な形で定義できる（3引き数のmin, maxは3番目の引き数に最適値を返す）．ここで述語Diagram(a, n1, n2)は，工程aがノードn1からノードn2へのアークであることを意味している．

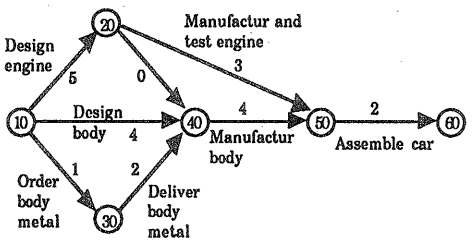


図3　PERT線図

```
Earliest_Start_Time(A : Es):-
    Diagram(A, N, _),
    max(Es1 + D, (
        Diagram(B, _, N),
        Duration(B : D),
        Earliest_Start_Time(B : Es1)
    ), Es),!.
Earliest_Start_Time(A : 0).

Latest_Finish_Time(A : Lf):-
    Diagram(A, _, N),
    min(Lf1 - D, (
        Diagram(B, N, _),
        Duration(B : D),
        Latest_Finish_Time(B : Lf1)
    ), Lf),!.
Latest_Finish_Time(A : Es + D):-
    Earliest_Start_Time(A : Es),
    Duration(A : D).
```

最早開始時間や最遅終了時間は，単に時間の制約のみを考慮したものである．しかし，現実の問題では，使用可能な資源（機械，電力，人力など）に制約があり，これらを考慮して各工程の実行スケジュールを作る必要がある．これは，Keyed CLPでつぎのようにモデル化できる．述語Scheduleは時刻Timeにおいて時間の制約のみを考慮した実行可能スケジュールを作り出す．結果は，キー付き述語Activeに保存される．Active(a, t : on)は工程aが時刻tにおいて実行中であることを意味する．このように，結果をキー付き述語に保存することにより，その値を述語の引き数として持たせなくて済むことになる．作り出されたスケジュールをもとに，各資源の必要量が計算され，使用可能な量の上限と比較される．もし制約を満たさなければバックトラックが起こり，述語Schedulingは別のスケジューリングを行なう．もし得られたスケジュールが資源制約を満たせば，つぎの時刻のスケジューリングに移る．

```
Resource_Analysis(Time):-
    Scheduling(Time),
    Resource_Needs(Time, engineers : A),
    Resource_Limit(engineers : A1),
    A <= A1,
    Resource_Needs(Time, power_units : B),
    Resource_Limit(power_units: B1),
    B <= B1,
    Resource_Needs(Time, draftsmen : C),
    Resource_Limit(draftsmen : C1),
    C <= C1,
    (Active(end, Time : on) ;
        resource_analysis1(Time + 1)),!.

Resource_Needs(Time, Res : Needs):-
    sum(R, (
        Diagram(A, _, _),
        Active(A, Time : on),
        Duration(A : D),
        Resources(A, Res : R)
        ),
    Needs),!.
Resource_Needs(Time, Res : 0).
```

最後にプログラムの実行例を示す．各時刻において実行される工程および使用している資源を順に表示している．

```
| ?- analysis.
Week  0 :
Active Processes = [start, Design_Engine, Ord
er_Body_Metal]
Engineers  =  1
Power Units =  0
```

```
Draftsmen = 2

Week 1:
Active Processes = [Design_Engine, Deliver_
Body_Metal]
Engineers = 1
Power Units = 5
Draftsmen = 2

Week 2:
Active Processes = [Design_Engine, Deliver_
Body_Metal]
Engineers = 1
Power Units = 5
Draftsmen = 2

Week 3:
Active Processes = [Design_Engine]
Engineers = 1
Power Units = 0
Draftsmen = 2
...
```

### 3.3 多目的決定問題

複数の目的関数が与えられているような意思決定問題を多目的決定問題(multi-objective decision making problem)とよぶ．[Dhar 88]で取り上げられている問題を例にして，Keyed CLPによる多目的決定問題の解法について説明する．この問題は計算機の構成を決定するものである．計算機ハードウェア，空調設備，オペレーティングシステム(OS)，データベース管理システム(DBMS)のそれぞれに選択すべき候補が何種類か存在し，その中から制約条件を満たし，かつ，与えられた目的関数を最適化するような構成を選ぶことが目標となる．それぞれの選択には，つぎの2種類の制約が存在する．

A. 定量的な制約 (quantative constraints):
規則A-1：OSおよびDBMSが必要とするメモリを差し引いた残りは，少なくとも6MB必要である．
規則A-2：総コストは$375,000以下でなければならない．

```
quantative_constraints :-
    selection(: HW, AC, OS, DB, N),
    HW_Spec(HW : HWcost, HWmips,
        HWmem),
    AC_Spec(AC : ACcost, ACpower, ACspace),
    OS_Spec(OS : OScost, OSmem),
    DB_Spec(DB : DBcost, DBmem, DBrec),

    /* Rule A-1 */
    HWmem - OSmem - DBmem/1024 >= 6.0,
    /* Rule A- 2 */
    HWcost + ACcost + OScost + N * DBcost
    <= 375000.
```

ここで，selectionは，ハードウェア，空調，OS，DBMSおよびその個数の選択を行なうキー付き述語である．それぞれの仕様は，HW_Spec，AC_Spec，OS_Spec，DB_Specに記述されている．

B. 定性的な制約 (qualitative constraints)
規則B-1：ハードウェアがIBMならば，OSはCMSでなければならない．
規則B-2：ハードウェアがIBMならば，空調の容量は400KW以上必要である．
規則B-3：ハードウェアがDECでOSがUNIXならば，DBMSはINGRESである．
規則B-4：OSがIMSならば，ハードウェアはIBMである．
規則B-5：DBMSがINGRESならば，ハードウェアにCDCは選べない．
規則B-6：5MIPS以下の計算機には，VMSをOSとして使用すべきではない．

```
qualitative_constraints :-
    selection(: HW, AC, OS, DB, N),
    HW_Spec(HW : HWcost, HWmips,
        HWmem),
    AC_Spec(AC : ACcost, ACpower, ACspace),
    OS_Spec(OS : OScost, OSmem),
    DB_Spec(DB : DBcost, DBmem, DBrec),

    /* Rule B-1 */
    IF(EQ(HW, 'IBM'), EQ(OS, 'CMS')),
    /* Rule B-2*/
    IF(EQ(HW, 'IBM'), ACpower > 400 ),
    /* Rule B-3 */
    IF (EQ(HW, 'DEC'), EQ(OS, 'UNIX')),
        EQ(DB, 'INGRES') ),
    /* Rule B-4 */
    IF(EQ(OS, 'IMS'), EQ(HW, 'IBM')),
    /* Rule B-5 */
    IF(EQ(DB, 'INGRES'),
        NEQ(HW, 'CDC') ),
    /* Rule B-6 */
    IF(HWmips < 5, NEQ(OS, 'VMS') ).
```

ここで，述語IFはつぎのように定義されている．

```
IF(Cond, Then) :- call(Cond),!,call(Then).
IF(_,_).
```

また，EQは単一化可能なとき，NEQは単一化できないときに成功する述語である．この問題の実行可能解はつぎのように定義される．

```
Feasible(HW, AC, OS, DB, N, TotCost,
        HWmips):-
    selection(: HW, AC, OS, DB, N),
    quantitative_constraints,
    qualitative_constraints
    TotalCost(:TotCost),
    HW_Spec(HW : HWcost, HWmips,
        HWmem).
```

コスト最小の組み合わせを選ぶには，つぎのゴールを実行する．

```
|?- min(TotCost, Feasible(HW, AC, OS, DB,
N,  TotCost, HWmips)).
```

同様に，計算機のMIPS値が最大の組み合わせを求めるには，つぎのゴールを実行する．

```
|?- max(HWmips, Feasible(HW, AC, OS, DB,
 N, TotCost, HWmips)).
```

２つの目標に対する最適解が異なる場合は，この２つの目標の組み合わせて評価を行う必要がある．ここでは，目的関数の間に優先順位をつける方法を使う．まずMIPS値を優先した選択を行なうには，つぎのゴールを実行する．述語min, maxはネストさせることができる．

```
|?- min(TotCost, max(HWmips, Feasible(
HW,  AC, OS, DB, N, TotCost, HWmips))).
```

これは，MIPS値が最大となるような計算機の構成の中でコストを最小にするものを求めるものである．逆に，コストを優先させた選択をするには，つぎのゴールを実行する．

```
|?- max(HWmips, min(TotCost, Feasible(
HW, AC, OS, DB, N, TotCost, HWmips))).
```

このように，Keyed CLPは優先順位がついた複数の目的関数が存在するような多目的決定問題も扱うことが可能である．

## 4. 考察

パーソナルコンピュータで広く普及している問題解決のためのツールとして，表計算言語(Spread Sheet)がある．表計算言語は，基本的には，２次元の表，および，表の各セル(cell)内に記述された数値，制約式により問題が記述される．制約式といっても実際には代入式であり，計算の方向が定まっている．また，代入関係が循環してはならないという制限も存在する．したがって，代数的に解が存在する場合でも，表計算言語では求められないといった場合がある．また各セルの指定はその座標（X座標，Y座標）によって行なわれるので，表示の上では表の形式をとっているが，実際の処理には表のもつ位相的な構造は必ずしも反映されない．しかし，表を関係表(relational table)として見ることにより，より簡潔な形で問題の記述ができる．

例として，表１のスプレッドシートを考える．

表１　スプレッドシート

| | A | B | C | D |
|---|---|---|---|---|
| 1 | 品名 | 単価 | 個数 | 金額 |
| 2 | パソコン本体 | 298,000 | 2 | B2 * C2 |
| 3 | ディスプレイ | 99,800 | 2 | B3 * C3 |
| 4 | プリンタ | 324,000 | 1 | B4 * C4 |
| 5 | ワープロソフト | 86,000 | 2 | B5 * C5 |
| 6 | | | | |
| 7 | 合計 | | | SUM(D2:D5) |

この表に対応するKeyed CLPプログラムはつぎのようになる．

```
goods(パソコン本体).
goods(カラーディスプレイ).
goods(プリンタ).
goods(ワープロソフト).
price(パソコン本体 : 298000).
price(カラーディスプレイ : 99800).
price(プリンタ : 324000).
price(ワープロソフト : 86000).
number_of_purchases(パソコン本体 : 2).
number_of_purchases(カラーディスプレイ : 2).
number_of_purchases(プリンタ : 1).
number_of_purchases(ワープロソフト : 2).
payment(G : P * N) :-
    price(G : P), number_of_purchases(G : N).
total_payment(S):-
    sum(X, (goods(G), payment(G : X)), S).
```

単価，購入個数，金額は品名が決まると一意に決定されるので，品名をキー属性としたキー付き述語で表現することができる．さらに，購入個数をつぎのように変数に置き換えることにより，予算内で購入可能なパソコンのセット数を計算することもできる．

```
number_of_purchases(パソコン本体 : X).
number_of_purchases(ディスプレイ:X):-
        number_of_purchases(パソコン本体 : X).
number_of_purchases(ワープロソフト : X):-
        number_of_purchases(パソコン本体 : X).
```

予算200万円以内で買えるセット数はつぎのように求められる．

```
|?- number_of_purchases(パソコン本体 : X),
 total_payment(2000000).
X = 3.46
```

したがって，3セットまで買えることになる．このような，双方向の計算が可能という点は，制約論理プログラミングの特長である．

Keyed CLPによる問題の記述は，表計算言語の抽象化として見ることもできる．各述語は，関係の内包的記述，すなわち，関係に含まれる要素を外延的に並べるのではなく，関係の要素が満たすべき制約条件を記述したもの，であると考えられる．その制約条件は関係のすべての要素に対して共通であり，表計算言語のように各セルごとに式を記述する必要はない†．また，制約条件の変更，表へのタプルの追加などに対し，柔軟に対応することができる．さらに，表計算言語では，値の取り出しがその座標によって行なわれるのに対し，論理プログラミング言語では，述語のパターンマッチにより行なわれる．これは関係表の検索に相当する．

Keyed CLPはまた，線形計画問題のモデル化を目的の1つとしている．対象がもつ制約を，直接数式の形で取り出すことは，問題の規模が大きくなると容易ではない．また，問題のモデルの再利用 – たとえば，ある問題のパラメータ，サイズを変えただけの問題の解法に，すでに作成した問題のモデルを利用する等 – の面から見ても好ましくない．ユーザーが直接数式を記述することなしに線形計画問題を生成するための専用のシステムもいくつか提案されている[Murphy 86] [Krishnan 90]．Keyed CLPでは，問題の記述は個々の構成要素がもつ断片的な制約の集まりにより行われ，システムはその中から線形制約を取り出し，線形計画問題を解いてくれる．記述されたプログラムは，人間が見ても理解可能であり，また計算機にとっても直接解釈／実行することができるものとなっている．したがって，Keyed CLPをユーザーの記述から線形計画問題を生成するLP Formulatorとして見ることができる．

† 最近では，Lotus Improv (Lotus Development Corporation)のように，関係の概念を利用した表計算言語も発表されている．

## 参考文献


[Aiba 88] A.Aiba, K.Sakai, Y.Sato et al., Constraint Logic Programming Language CAL, Proc. of the Int. Conf. on FGCS 1988, 252/262 (1988).

[Dhar 88] Vasant Dhar and Albert Croker, Knowledge Bases Decision Support in Business : Issues and a Solution, IEEE Expert, Spring, 53/62 (1988).

[Dincbas 88] M.Dincbas, P.Van Hentenryck et al., The Constraint Logic Programming Language CHIP, Proc. of the Int. Conf. on FGCS 1988, 693/702 (1988).

[Dolk 86] D.Dolk, Data as Models: An Approach to Implementing Model Management, Decision Support Systems 2, 73/80 (1986).

[Geoffrion 87] A. Geoffrion, An Introduction to Structured Modeling, Management Science, 33-5, 547/588 (1987).

[Jaffer 87] J.Jaffer and J.Lassez, Constraint Logic Programming, in Proc. of POPL-87, Munich, (1987).

[Krishnan 90] R.Krishnan, A Logic Modeling Language for Automated Model Construction, Decision Support Systems 6, 123/152 (1990).

[Murphy 86] F. Murphy and E Stohr, An Intelligent System for Formulating Linear Programs, Decision Support Systems, Vol.2, 39/47 (1986).

[Willis 87] R.Willis, Computer Models for Business Decisions, John Wiley & Sons (1987).

## 付録1　生産計画のモデル

```
/* 製品 */
Goods(a).
Goods(b).
Goods(c).
Goods(d).

/* 生産量 */
Product(a : X):- X >= 0.
Product(b : Y):- Y >= 0.
Product(c : Z):- Z >= 0.
Product(d : Z):- Z >= 0.

/* 資源の種類 */
Resource(materials).
Resource(epower).
Resource(mpower).

/* 一単位の製品の生産に必要な資源 */
Need(materials, a : 2.5).
Need(materials, b : 5).
Need(materials, c : 6).
Need(materials, d : 2).

Need(epower, a : 5).
Need(epower, b : 6).
Need(epower, c : 7).
Need(epower, d : 3).

Need(mpower, a : 3).
Need(mpower, b : 2).
Need(mpower, c : 2).
Need(mpower, d : 5).

/* 使用可能な資源 */
Resource_Available(materials : 350).
Resource_Available(epower : 450).
Resource_Available(mpower : 240).

/*  一単位の製品の生産による利益 */
Profit(a : 9).
Profit(b : 5).
Profit(c : 8).
Profit(d : 11).

/* 総利益 */
Total_Profit(S):-
    sum(X * Y,
        (Goods(P), Product(P : X), Profit(P : Y)),
     S).

/* 資源制約 */
Resource_Constr(R):-
    sum(X * Y,
        (Resource(R), Goods(P),
        Need(R, P : X), Product(P : Y)),
    Z),
    Resource_Available(R : L), Z <= L, !.

/* 実行可能解 */
Feasible(S, Pa, Pb, Pc, Pd):-
    Product(a : Pa),
    Product(b : Pb),
    Product(c : Pc),
    Product(d : Pd),
    Pa > 25, Pb + Pc + Pd >= 30,
    (Pb = 0 ; Pc = 0 ; Pd = 0),
    all((Resource(R), Resource_Constr(R))),
    Total_Profit(S).
```