Keyed CLP : Features

©Kunihiko Hiraishi
School of Information Science,
Japan Advanced Institute of Science and Technology

1 Special Features of Keyed CLP

1.1 Keyed Predicates

In addition to ordinary predicates, Keyed CLP has an another type of predicates, called *keyed predicate*, which has the form

Predicate-Name(Key_1 ,...,Key_n : Arg_1 ,..., Arg_m).

In logic programming languages, each predicate can be seen as a relation. The word *key* has the same meaning as in the relational database theory, and represents a functional dependency in the relation. Each keyed predicate corresponds to a tuple with Key_1 ,..., Key_n as the key attributes. In a relation, a tuple is uniquely determined if all the values of the key attributes are specified. This property is preserved in the execution of programs, i.e. every predicate which has the same predicate name and the same key value must have the same value.

The fastest way to understand how the keyed predicate works is to see examples.

Example 1.

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

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

*** yes ***
```

p(X:A) is a keyed predicate and X is the key of this predicate. Execution of Keyed CLP is the same as ordinary prolog, excepting keyed predicates. In the above program, q(3,Z) is evaluated first and constraints X=3 and Z=3+Y are generated. Next p(k:3) (keyed predicate which has k as the key attribute) is evaluated, and at this time the value of predicate p with the key value k is determined as p(k:3). Therefore, the next goal p(k:Y) is unified with p(k:3) and Z becomes to 3+3=6. In this case, this keyed predicate works as a global variable.

Example 2.

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

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

*** yes ***
```

The values kept by a keyed predicate are not restricted to constants. In Example 2, by the unification between p(k:A,2) and p(X:A,B):-A>=B, the value of predicate p with key k is determined as p(k:A,2) with a constraint A≥2. a(A1) first gets a value A1=1 and then p(k:1,_) fails because 1≥2 does not holds. Next a(A1) gets a value A1=2 and then p(k:2,_) succeeds. Therefore only B = 2 is the solution.

1.2 Operational Model of Keyed CLP

In this report the word constraint is used as numerical constraints. The following constraints are called primitive constraints:

$$Expr = Expr,\ Expr \geq Expr,\ Expr \leq Expr,$$

$$Expr > Expr,\ Expr < Expr$$

where *Expr* is composed of variables, constants, +, -, * and /. Domain of variables is the real numbers. A Keyed CLP program consists of a finite number of rules, each of which has the form:

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

where c_i ($i = 1, \ldots, n$) are primitive constraints and A_j ($j = 1,...,i$) are terms. At each step of an execution, the goals has the form

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

where c_i ($i = 1,..., q$) are primitive constraints, A_j ($j = 1,...,r$) are terms, θ is a substitution and S is the stack for keyed predicate. Initially S is empty. Each A_j is called *a subgoal*. Let G_1 be a nonempty goal including a set C_1 of primitive constraints, which has the form

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

When C_1 is solvable, a new goal G_2 is derived as follows :

(i) A_1 is not a keyed predicate.

Suppose that the program P contains the following rule.

$$B_0 \text{ :- } C_2, D_1, D_2, \ldots, D_h.$$

where $A_1\theta$ and B_0 have a mgu θ'. Then the new goal is

$$G_2 = (C_1, C_2, [A_1\theta = B_0] \ ; D_1, D_2, \ldots, D_h, A_2, \ldots, A_m \ ; \theta \cup \theta' ; S).$$

New constraint is obtained by the union of C_1, C_2 and the constraints $[A_1\theta = B_0]$ generated by the unification of $A_1\theta$ and B_0. New substitution is the union of θ and θ'. Constraint $(C_1, C_2, [A_1\theta = B_0])$ must be solvable. The set S of keyed predicate stack does not change.

(ii) A_1 is a keyed predicate

$A_1\theta$ should not contain any variables in the key arguments. There are two cases :

a. S contains a keyed predicate D with the same predicate name and the same key value as $A_1\theta$.

$$G_2 = (C_1, [A_1\theta = B_0] \ ; A_2, \ldots, A_m \ ; \theta \cup \theta' ; S)$$

where $A_1\theta$ and D have a mgu θ'. In addition, $(C_1, [A_1\theta = B_0])$ is solvable. If $(C_1, [A_1\theta = B_0])$ are not solvable, the subgoal A_1 fails.

b. S does not contain any predicate D with the same predicate name and the same key value as $A_1\theta$.

Suppose that the program P contains a rule like

$$B_0 \text{ :- } C_2, D_1, D_2, \ldots, D_h.$$

where $A_1\theta$ and B_0 have a mgu θ'. Then

$$G_2 = (C_1, C_2, [A_1\theta = B_0] \ ; D_1, D_2, \ldots D_h, A_2, \ldots, A_m \ ; \theta \cup \theta' ; S \cup \{ A_1\theta \})$$

New constraint is obtained by the union of C_1, C_2 and the constraint $[A_1\theta = B_0]$ generated by the unification of $A_1\theta$ and B_0. New substitution is the union of θ and θ'. Constraint $(C_1, C_2, [A_1\theta = B_0])$ must be solvable. New keyed predicate stack is the union of S and $\{ A_1\theta \}$.

If such G_2 exists we say that there is *a derivation step* from G_1 to G_2. *A derivation sequence* is a (possibly infinite) sequence of goals wherein there is a derivation step to each goal from the preceding goal. A derivation sequence is successful if it is finite and its last goal contains only constraints and the keyed predicate stack. There exists nondeterminism on getting the next rule. Current implementation of Keyed CLP uses the same rule as Prolog. In a successful sequence, arguments of every predicate which has the same key value takes the same value.

We demonstrate the derivation sequence in Example 2.

Step 1. G_0 = (∅; q(2, B) ; ∅ ; ∅)

Step 2. Unify with a rule q(Z,A1):-p(k:A,Z),a(A1),p(k:A1,_).

G_1 = (Z = 2, B = A1 ; p(k : A, Z), a(A1), p(k : A1, _) ; ∅ ; ∅)

Step 3. Unify with a rule p(X:A,B):-A>=B.

G_2 = (Z = 2, B = A1, A = A', Z = B', A' >= B' ; a(A1), p(k : A1, _) ; k/X ; {p(k : A', B')})

Step 4. Unify with a rule a(1).

G_3 = (Z = 2, B = A1, A = A', Z = B', A' >= B', A1 = 1 ; p(k : A1, _) ; k/X ; {p(k : A', B')})

Step 5. Unify with a rule p(k : A',B') in the stack.

Z = 2, B = A1, A = A', Z = B', A' >= B', A1 = 1 , A1 = A' is not solvable.

Step 6. Unify with a rule a(2).

G_3 = (Z = 2, B = A1, A = A', Z = B', A' >= B', A1 = 2 ; p(k : A1, _) ; k/X ; {p(k : A', B')})

Step 7. Unify with a rule p(k : A',B') in the stack.

G_4 = (Z = 2, B = A1, A = A', Z = B', A' >= B', A1 = 2, A1 = A' ; ∅; k/X ; {p(k : A', B')})

Z = 2, B = A1, A = A', Z = B', A' >= B', A1 = 2, A1 = A' is solvable.

1.3 How does the Keyed Predicates Work?

Keyed predicates are used as global variables, each value in which is specified by key attributes. Using keyed predicates, we can keep values calculated by some process and use the value in another processes. Moreover, when we describe a program, we can reduce the number of arguments in each predicate by using keyed predicate.

Values calculated by some process are often referred by another processes. For example, when we intend to build a budget plan, the following values are usually used as basic data and are often referred when other values are calculated : costs of manufacturing goods, cost of materials and number of selling and so on. If we describe these values as a non-keyed predicate like sales_units(a,X):-..., the body part will be evaluated whenever sales_units is called. However, since the value of selling number is uniquely determined if the goods name is given, once the value is calculated, we have only to refer the value and recalculation is not necessary. This problem is avoidable by using keyed predicate with the goods name as the key attribute, i.e. sales_units(a:X):-....

We will show an example in which keyed predicates works for improving efficiency in computation. The following program calculates the Fibonacci numbers.

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

Let us consider a goal fib(5:X). fib(5:X) is determined by fib(4:X1) and fib(3:X2). fib(4:X1) is determined by fib(3:X1) and fib(2:X2). Fig. 1 show the derivation process for this goal.

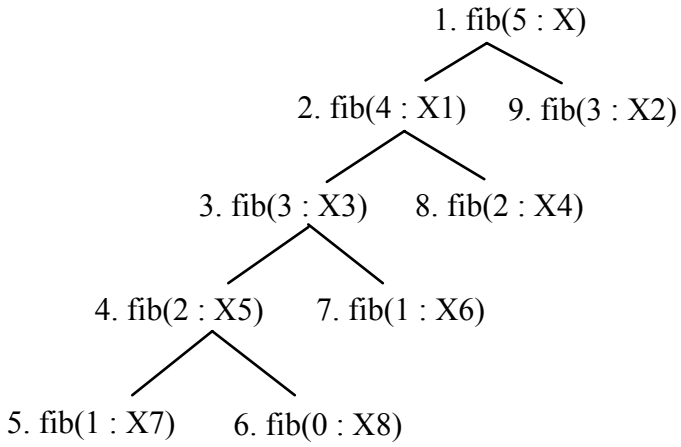

Fig. 1 Execution of fib(5 : X)

The number associated with each node of the tree represents the ordering in the execution in Keyed CLP. For keyed predicates in the nodes 7, 8 and 9, there exists keyed predicates having the same key value in 5, 4 and 3, respectively. Therefore evaluation of the body part of these predicate 7, 8, and 9 are omitted.

1.4 Nonlinear Constraints

In each derivation step, the simplex method is used for checking whether the new constraint is consistent with existing ones. If the constraint contains a nonlinear term, then it is replaced with a variable and the nonlinear term is separated from the linear constraint. Evaluation of each nonlinear constraint is delayed until it becomes linear by substituting variables and/or by solving other constraints. When the derivation step is successful, Keyed CLP interpreter outputs the substitutions to the variables in the initial goal and remaining (linear and nonlinear) constraints.

Basically, Keyed CLP solves only linear constraints. However, since the target domain of Keyed CLP is problems in OR/MS, linear constraint solver will cover most problem areas. In addition, it is essential to handle inequalities when practical problems are considered. Because, each value usually has some allowable range. It is not easy to handle inequalities when we solve nonlinear constraints.

1.5 Higher Order Predicates

Keyed CLP has the following higher order predicates for solving linear optimization problem.

min(*Expr*, *Goal*),

max(*Expr*, *Goal*)

where *Expr* is a linear formula. The predicate min (max) find the minimum (maximum) value of *Expr* satisfying the *Goal*. If the Goal has alternatives, then min calculate every optimum value for each alternative and then find the optimum value among them. Even when the region represented by given constraints is the union of some convex areas, the predicate min (max) works correctly. *Branch and bound* method is used for

searching the optimum value. We demonstrate how min (max) finds the optimum value by the following example.

Example 3.

```
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 ***
```

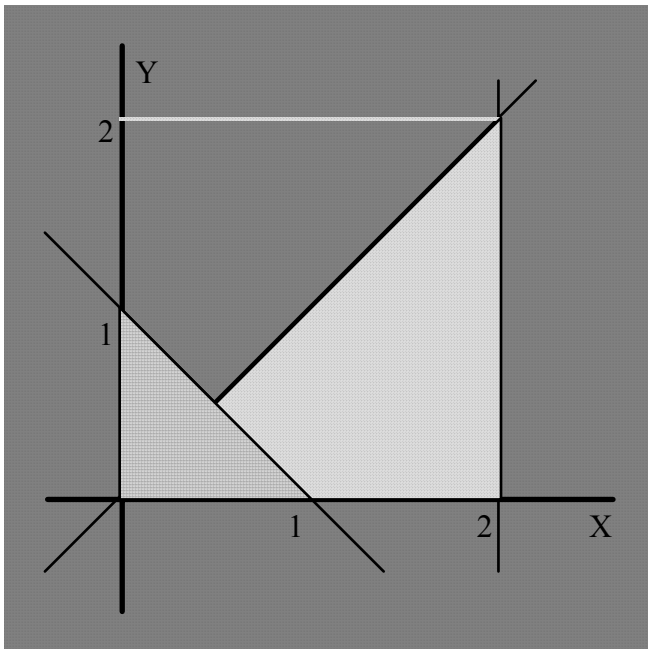

Fig. 2 Union of Two Convex Sets

The region represented by the predicate region(X, Y) is the union of two convex sets (Fig. 2). The predicate max find the maximum value of Y for each region (the maximum value for the first set is Y=2 and Y=1 for the second set), and then take the maximum value Y=2. The optimum value for previous alternatives are used for the upper (lower) bound of the function when another alternatives are searched. At some derivation step, if the new alternative will not generate a value which is better than the bound, then the derivation step will stop at the level and backtracking will occur. We note that the second argument of min (max) can contain min/max. This allow us to solve the min-max (max-min) type optimization problems.

In addition to min and max, Keyed CLP has the following higher order predicates.

- sum(*Expr*, *Goal*, *Res*) : unify the summation of the value of *Expr* for every alternatives of *Goal* with the variable *Res*;
- avg(*Expr*, *Goal*, *Res*) : unify the average of the value of *Expr* for every alternatives of *Goal* with the variable *Res*;
- count(*Goal*, *Res*) : unify the number of alternatives of *Goal* with the variable *Res*;
- all(*Goal*) : find all alternatives.

The predicate sum works when *Expr* contains variables. In this case, sum generate an equality like *Res* = $Expr_1 + Expr_2 + ... + Expr_n$. For example, let us consider the following program.

Example 4.

```
key(k1).
key(k2).
key(k3).
a(k1 : X1).
a(k2 : 2 * X3 + 5):- a(k3 : X3).
a(k3 : X1 + X2):- a(k1 : X1), a(k2 : X2).
goal(S):- sum(X, (key(K), a(K : X)), S).

| ?- goal(S), a(k1 : 3).
S = -16.0

*** yes ***
```

By the evaluation of goal(S), the following equalities are generated :

```
X2 = 2 * X3 + 5
X3 = X1 + X2
S = X1 + X2 + X3
```

X1=3 is given by the second subgoal a(k1 : 3) and therefore we obtain X2=-11, X3=-8, S=-16 by solving the above equalities. In ordinal Prolog, the predicate all usually implemented as follows :

```
all(X):- call(X), fail.
all(_).
```

The above all find all alternatives by backtracking. The predicate all in Keyed CLP works in a different way. It preserves the history of execution and the generated values can be passed through keyed predicates. In the following program, key(K) has three alternatives and for each alternative key(K) (K = 0, 1, 2), a keyed predicate p(K:K+1) is added to the stack. These values are preserved even after the evaluation of the predicate all. Therefore, X=1, Y=2, Z=3 is obtained.

Example 5.

```
key(0).
key(1).
key(2).
p(K : K + 1).
q(X + Y + Z):-
      all((key(K), p(K : V)),
      p(0 : X), p(1 : Y), p(2 : Z).

| ?- q(A).
A = 6

*** yes ***
```

1.6 Keyed CLP Interpreter

Current version of Keyed CLP interpreter is developed by C language on UNIX operating system. Fig. 3 shows an overview of the system. User's queries are processed by the Engine Program control for higher order predicates and keyed predicates are also processed by the Engine. Numerical constraints are separated into linear constraints and nonliner constraints. Linear constraints are solved by the Simplex Solver. Evaluations of nonlinear constraints are delayed until they become linear. Built-in function are processes by the Built-in Function Module and their eveluations are delayed until every argument in the function is constant. Built-in functions which have variables in their arguments are also stored in the Nonlinear Constraint Stack.

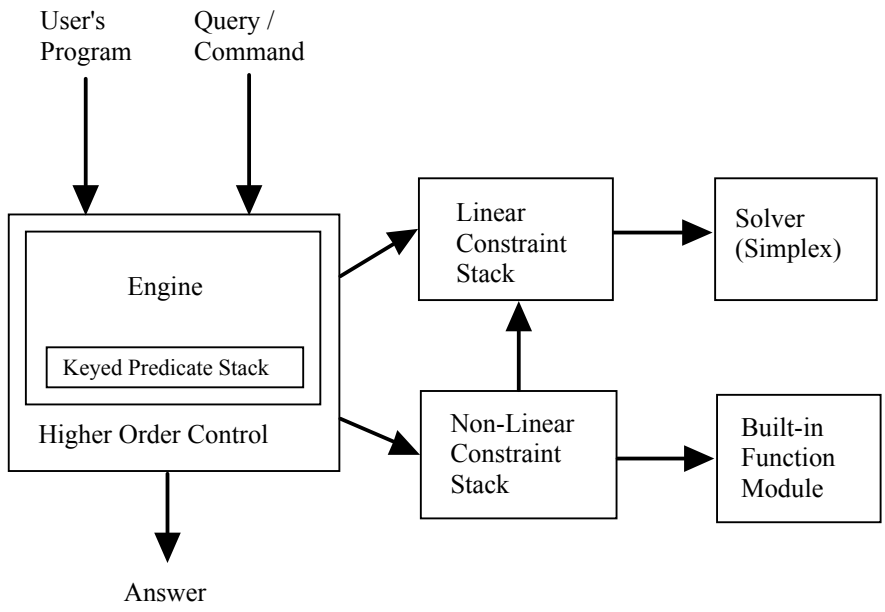

Fig. 3 Keyed CLP Interpreter - System Overview -

2 Problem Solving using Keyed CLP

2.1 Production Planning

The program in Appendix I represents a linear programming problem for deciding production plan in a small company. The company produces four kinds of products a, b, c, d. For each product, three kinds of resources (material, electric power and man power) necessary for production is given. For each resources, upper limit of available quantity is given. One of the aim is to find quantity of production for each product which maximize the total profit and satisfies the resource constraints. In addition, the predicate in the program contains the following nonlinear constraint:

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

where Pb, Pc and Pd are quantity of production for each product b, c, d, respectively. The meaning of this constraint is that at least one of the three products should not be produced. For such constraints, the predicate max easily find the optimal solution.

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

*** yes ***
```

In this program, the predicate Products(:Pa,Pb,Pc,Pd) works as a place to store global variables. This predicate has no key attribute. This means that it has a unique value in a derivation sequence. When Products(:Pa,Pb,Pc,Pd) is called at the first time, inequalities Pa>=0, Pb>=0, Pc>=0, Pd>=0 are evaluated. At the next time Products(:Pa,Pb,Pc,Pd) is called, Products(:Pa,Pb,Pc,Pd) is already exists in the keyed predicate stack and therefore the inequalities in the body are not evaluated.

2.3 PERT Diagram and Scheduling

Next problem is a scheduling problem using PERT (Program Evaluation and Review Technique).

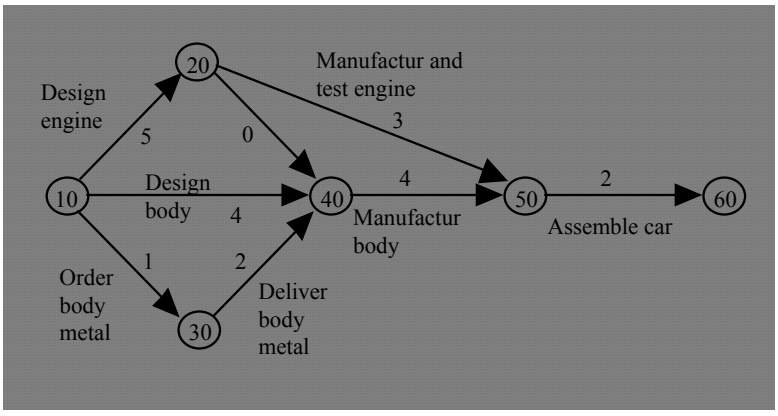

Fig.4 PERT Diagram for Manufacturing Cars

The PERT diagram in Fig. 4 represents processes of manufacturing new cars [Willis 87]. A number associated with each arc is the duration of the activity. The arc from node 20 to 40 has 0 as the duration because it is a dummy arc which represent only the precedence relation. Using a PERT diagram, the earliest start time and the latest finish time are calculated for each node, and find critical activities[1]. The earliest start time for each activity is found by taking the maximum of the earliest start time + the duration for each of its preceding activities. The latest finish time for each activity is found by taking the minimum of the latest finish time - the duration for each of its successive activities. Using the build-in predicate max, the earliest start time and the latest finish time for each node can be easily written as follows (the whole program is in Appendix II) :

```
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).
```

In this program, each predicate Diagram(a, n1, n2) means that an activity a is represented by an arc from a node n1 to a node n2. The following is a result of the execution of the predicate time_analysis, which calculates the earliest start time, the earliest finish time, the latest start time, the latest finish time and the total float for each activity.

```
| ?- time_analysis.
Activity : start
      Earliest Start Time  =   0
      Earliest Finish Time =   0
      Latest Start Time    =   0
      Latest Finish Time   =   0
      Total Float         =   0

Activity : Design_Engine
      Earliest Start Time  =   0
      Earliest Finish Time =   5
      Latest Start Time    =   0
      Latest Finish Time   =   5
      Total Float         =   0

Activity : Design_Body
      Earliest Start Time  =   0
      Earliest Finish Time =   4
      Latest Start Time    =   1
      Latest Finish Time   =   5
      Total Float         =   1

Activity : Order_Body_Metal
```

[1] Critical activity means that delay of a critical activity causes delay of the whole system.

```
      Earliest Start Time  =   0
      Earliest Finish Time =   1
      Latest Start Time    =   2
      Latest Finish Time   =   3
      Total Float          =   2

Activity : Deliver_Body_Metal
      Earliest Start Time  =   1
      Earliest Finish Time =   3
      Latest Start Time    =   3
      Latest Finish Time   =   5
      Total Float          =   2

Activity : dummy
      Earliest Start Time  =   5
      Earliest Finish Time =   5
      Latest Start Time    =   5
      Latest Finish Time   =   5
      Total Float          =   0

Activity : Manufacture_Body
      Earliest Start Time  =   5
      Earliest Finish Time =   9
      Latest Start Time    =   5
      Latest Finish Time   =   9
      Total Float          =   0

Activity : Manufacture_and_Test_Engine
      Earliest Start Time  =   5
      Earliest Finish Time =   8
      Latest Start Time    =   6
      Latest Finish Time   =   9
      Total Float          =   1

Activity : Assemble_Car
      Earliest Start Time  =   9
      Earliest Finish Time =  11
      Latest Start Time    =   9
      Latest Finish Time   =  11
      Total Float          =   0

Activity : end
      Earliest Start Time  =  11
      Earliest Finish Time =  11
      Latest Start Time    =  11
      Latest Finish Time   =  11
      Total Float          =   0


*** yes ***

Retry? n

Interrupted.
```

Earliest start time and latest finish time are obtained by considering only time constraints. In practical problems, however, there exists some limitations for available resources (such as machines, electric power and man power), and therefore we have to make a schedule satisfying the limitations. The predicate Schedule generates successively feasible schedules considering only time constraints. The results are stored in the predicate Active. Active(a,t:on) means that an activity a is processes at a time t. Using this scheduling, necessary resources (engineers, power units and draftsmen) are calculated, and are compared with the upper limits of available resources. If at least one of the resources does not satisfy the constraint, backtracking is

occurred and the predicate Schedule generate an another schedule. If the schedule satisfies all of the resource constraint, the program shifts to the scheduling of the next time.

```
resource_analysis1(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,
      resource_analysis_summary(Time, A, B, C),
      (Active(end, Time : on),
            nl, write("Project Duration = "),
            print("%3.0f", Time), nl,
            Total_Cost(TC),
            write("Total Direct Cost(NPV) = "),
            print("%7.0f", TC),nl,
            dump([TC]) ;
            resource_analysis1(Time + 1)
      ),!.
```

We show the execution of this scheduling. At each time, executing activities and the amount of resources used are displayed.

```
| ?- resource_analysis.
Week   0 :
      Active Processes = [start, Design_Engine, Order_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

Week   4 :
      Active Processes = [Design_Engine]
      Engineers   =   1
      Power Units =   0
      Draftsmen   =   2

Week   5 :
      Active      Processes      =      [Design_Body,      dummy,
Manufacture_and_Test_Engine]
      Engineers   =   4
      Power Units =  20
```

```
      Draftsmen   =   2

Week   6 :
      Active Processes = [Design_Body, Manufacture_and_Test_Engine]
      Engineers   =   4
      Power Units =  20
      Draftsmen   =   2

Week   7 :
      Active Processes = [Design_Body, Manufacture_and_Test_Engine]
      Engineers   =   4
      Power Units =  20
      Draftsmen   =   2

Week   8 :
      Active Processes = [Design_Body]
      Engineers   =   1
      Power Units =   0
      Draftsmen   =   2

Week   9 :
      Active Processes = [Manufacture_Body]
      Engineers   =   2
      Power Units =  15
      Draftsmen   =   0

Week  10 :
      Active Processes = [Manufacture_Body]
      Engineers   =   2
      Power Units =  15
      Draftsmen   =   0

Week  11 :
      Active Processes = [Manufacture_Body]
      Engineers   =   2
      Power Units =  15
      Draftsmen   =   0

Week  12 :
      Active Processes = [Manufacture_Body]
      Engineers   =   2
      Power Units =  15
      Draftsmen   =   0

Week  13 :
      Active Processes = [Assemble_Car]
      Engineers   =   1
      Power Units =  15
      Draftsmen   =   0

Week  14 :
      Active Processes = [Assemble_Car]
      Engineers   =   1
      Power Units =  15
      Draftsmen   =   0

Week  15 :
      Active Processes = [end]
      Engineers   =   0
      Power Units =   0
      Draftsmen   =   0


Project Duration =  15
```

```
Total Direct Cost(NPV) =  137231

*** yes ***

Retry? n

Interrupted.
```

2.3 Multi-Objective Decision Making

A decision-making problem which has more than two objective functions are called *a multi-objective decision making problem*. We use a problem described in [Dhar 88]. This probvlem is for deciding a computer system to introduce, and is described as follows (the whole program is in Appendix III) :

Choice sets :

There are four choice sets: hardware, air-conditioning units, operating systems, and database management systems. Each alternative is defined over the set of choice-set attributes, e.g. hardware has four attributes : Cost, MIPS, Memory (M bytes) and Annual maintenance. This choice-set of hardware is written as the following Keyed CLP Program :

```
/*
        Hardware Specifications
        Name : Cost(K), MIPS, Memory(MB), Annual maintenance
*/

Hardware('DEC').
Hardware('IBM').
Hardware('CDC').
Hardware('ATT').

HW_Spec('DEC' : 300000, 5, 8, low   ).
HW_Spec('IBM' : 500000, 5, 8, high  ).
HW_Spec('CDC' : 350000, 5, 8, high  ).
HW_Spec('ATT' : 100000, 2, 4, medium).
```

There are two types of constraints.

Quantitative constraints :

Rule A1 : At least 6M bytes must be available apart from memory requirements of the operating system and the database system.

Rule A2 : The total cost of the various information system components should not exceed $375,000.

Qualitative constraints :

Rule B1 : If IBM is selected as the hardware, then the OS must be CMS.

Rule B2 : If IBM is selected as the hardware, then the AirConditioning power must be greater than 400 KW.

Rule B3 : If the Hardware is DEC and the OS is UNIX, then INGRES must be selected as the DBMS.

Rule B4 : If IMS is selected as the OS, then the hardware must be IBM.

Rule B5 : If INGRES is selected as the DBMS, then the hardware must not be CDS.

Rule B6 : VMS operating system should not be installed on hardware that is less than 5 MIPS.

These constraints are described by Keyed CLP program as follows :

```
constraints :-
```

```
      quantative_constraints,
      qualitative_constraints.

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

      /* Rule A1 */
      HWmem - OSmem - DBmem/1024 >= 6.0,

      /* Rule A2 */
      HWcost + ACcost + OScost + N * DBcost <= 375000.

qualitative_constraints :-

      selection(: HW, AC, OS, DB, N),

      HW_Spec(HW : HWcost, HWmips, HWmem, HWmaint),
      AC_Spec(AC : ACcost, ACpower, ACspace),
      OS_Spec(OS : OScost, OSmem),
      DB_Spec(DB : DBcost, DBmem, DBrec),

      /* Rule B1 */
      IF( EQ(HW,  'IBM'), EQ(OS, 'CMS') ),

      /* Rule B2 */
      IF( EQ(HW, 'IBM'), ACpower > 400 ),

      /* Rule B3 */
      IF( ( EQ(HW, 'DEC'), EQ(OS, 'UNIX') ),
            EQ(DB, 'INGRES') ),

      /* Rule B4 */
      IF( EQ(OS, 'IMS'), EQ(HW, 'IBM') ),

      /* Rule B5 */
      IF( EQ(DB, 'INGRES'), NEQ(HW, 'CDC') ),

      /* Rule B6 */
      IF( HWmips < 5, NEQ(OS, 'VMS') ).
```

where the predicate selection is a keyed predicate with empty key which works as global variables containing selected tuple from the choice sets. The predicate IF is defined by

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

EQ(X,Y) succeeds iff X and Y are unifirable and NEQ succeeds iff X and Y are not unifirable. Feasible solutions are written as follows :

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

go1 finds the solution that minimizes the total cost.

```
go1:-
      min(TotCost,
            Feasible(HW, AC, OS, DB, N, TotCost, HWmips)
```

```
        ),
        SpaceRequired(: Space),
        write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).
```

go2 finds the solution that maximizes MIPS value.

```
go2:-
        max(HWmips,
              Feasible(HW, AC, OS, DB, N, TotCost, HWmips)
        ),
        SpaceRequired(: Space),
        write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).
```

The aim of this problem is to find a solution optimizing both of the objectives, minimizing costs and maximizing MIPS value. We use here a technique of *sequential optimization*. This technique find the optimum value by assigning an order of evaluation among the objectives. go3 first find solutions that maximizes MIPS value, and among them find the solution that minimizes the total cost. go4 first find solutions that minimizes the total cost, and among them find the solution that maximizes MIPS value.

```
go3:-
        min(TotCost,
              max(HWmips,
                    Feasible(HW, AC, OS, DB, N,
                          TotCost, HWmips)
              )
        ),
        SpaceRequired(: Space),
        write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).

go4:-
        max(HWmips,
              min(TotCost,
                    Feasible(HW, AC, OS, DB, N,
                          TotCost, HWmips)
              )
        ),
        SpaceRequired(: Space),
        write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).
```

Nesting levels of min and max is only restricted by the available memory.

2.4 Keyed CLP As an Extension of Spread Sheets

Spread sheet are widely recognized as a tool of modeling and problem solving on personal computers. In spread sheets, problems are modeled on two-dimensional tables and constraints are described as formulas in each cell. Calculation of spread sheets is executed by substituting each cell with the calculated valued of the formula. There is a direction defined by referring other cells in each formula. Since this referring relation must be acyclic, spread sheets cannot always find the solution even when the solution is obtainable by algebraic operations. In addition, each cell is specified by its coordinate on the table like A10, B5. Therefore, though spread sheets have a form of tables, their topological structure as tables is not reflected on modeling processes[2]. If we treat a table as a relational table, we can model problems in a more sophisticated way. For example, let us consider the sheet in Table 1.

Table 1 An example of spread sheets

[2] Spread sheets have a function of copying foumulas with shifting coordinates. This might be a function using topological structure

	A	B	C	D	
1	Goods	Unit Price	Number	Total Price	
2	Computer	298,000	2	B2 * C2	
3	Monitor	99,800	2	B3 * C3	
4	Printer	324,000	1	B4 * C4	
5	Software	86,000	2	B5 * C5	
6					
7	Total			SUM(D2:D5)	

The following is a Keyed CLP program corresponding to the above sheet:

```
goods(computer).
goods(monitor).
goods(printer).
goods(software).

unit_price(computer : 298000).
unit_price(monitor : 99800).
unit_price(printer : 324000).
unit_price(software : 86000).

number_of_purchases(computer : 2).
number_of_purchases(monitor : 2).
number_of_purchases(printer : 1).
number_of_purchases(software : 2).

payment(G : P * N):-
      unit_price(G : P), number_of_purchases(G : N).

total_payment(S):- sum(X, (goods(G), payment(G : X)), S).
```

Unit price, number of purchases and payment are uniquely determined when a goods name is specified. Therefore, they can be expressed by keyed predicate having goods name as the key attribute. Moreover, by replacing constants in the predicate number_of_purchases with variables, the maximum number of computer sets we can purchase within a given total payment can be found..

```
number_of_purchases(computer : X).
number_of_purchases(monitor : X):-
      number_of_purchases(computer : X).
number_of_purchases(software : X):-
      number_of_purchases(computer : X).
```

Given a budget 2,000,000 yen, the number of computer sets is calculated by

```
| ?- number_of_purchases(computer : X),
total_payment(2000000).
X = 3.46

*** yes ***
```

Therefore, we can purchase three sets. Bi-directional computation is one of the characteristic features of constraint programming.
Problem solving using Keyed CLP can be seen as an abstraction of spread sheet type languages. Each predicate in Keyed CLP is *an intensional description* of a relation, i.e. a description of constraints satisfied by each element in the relation. This constraints are common to all the elements in the relation. Therefore, we do not have to define a formula in every cell in the sheet as in spread sheets. Moreover, while each value is specified by the coordinate in spread sheets, it is specified by key attributes in Keyed CLP.

Appendix I

```
/*********************************************************************

      Linear Programming

*********************************************************************/

Goods(a).
Goods(b).
Goods(c).
Goods(d).

Product(a : X):- X >= 0.
Product(b : X):- X >= 0.
Product(c : X):- X >= 0.
Product(d : X):- X >= 0.

Resource(materials).
Resource(epower).
Resource(mpower).

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).

Profit(a : 9).
Profit(b : 5).
Profit(c : 8).
Profit(d : 11).

Res_Available(materials : 350).
Res_Available(epower : 450).
Res_Available(mpower : 240).

TotProfit(S):-
      sum(X * Y, (Goods(P), Product(P : X), Profit(P : Y)), S).

Decision_Vars(: Pa, Pb, Pc, Pd):-
      Product(a : Pa), Product(b : Pb), Product(c : Pc), Product(d : Pd).

Constr(R):-
      sum(X * Y,
            (Resource(R), Goods(P), Need(R, P : X), Product(P : Y)),
```

```
      Z),
      Res_Available(R : L),
      Z <= L.

constraints:-
      all((Resource(R), Constr(R))),
/*
      Constr(materials), Constr(epower), Constr(mpower),
*/
      Decision_Vars(: Pa, Pb, Pc, Pd),
      Pa >= 25, Pb + Pc + Pd >= 30, !, (Pb =  0 ; Pc =  0 ; Pd  = 0).

Feasible(S, Pa, Pb, Pc, Pd):-
      Decision_Vars(: Pa, Pb, Pc, Pd),
      TotProfit(S),
      constraints.

RANGE(V, X, L, U):- min(V, X, L), max(V, X, U).
```

Appendix II

```
/*

      Project Management and Control

      Robert J. Willis :
      Computer Models for Business Decisions,
      JOHN WILEY & SONS (1987).

*/

/*
      Time Analysis
*/


time_analysis:-
      Activities(: As),
      time_analysis1(As).

time_analysis1([]).
time_analysis1([A|R]):-
      Earliest_Start_Time(A : Es),
      Earliest_Finish_Time(A : Ef),
      Latest_Start_Time(A : Ls),
      Latest_Finish_Time(A : Lf),
      Total_Float(A : Tf),
      time_analysis_summary(A, Es, Ef, Ls, Lf, Tf),
      time_analysis1(R).

time_analysis_summary(A, Es, Ef, Ls, Lf, Tf):-
      write("Activity : "), write(A),nl,
      tab, write("Earliest Start Time  = "), print("%3.0Lf", Es), nl,
      tab, write("Earliest Finish Time = "), print("%3.0Lf", Ef), nl,
      tab, write("Latest Start Time    = "), print("%3.0Lf", Ls), nl,
      tab, write("Latest Finish Time   = "), print("%3.0Lf", Lf), nl,
      tab, write("Total Float         = "), print("%3.0Lf", Tf), nl, nl,
      dump([Es, Ef, Ls, Lf, Tf]).


/*
      Earliest Start Time :
      The earliest start time for each activity is found by
      taking the maximum of the earliest start time + duration
      for each of its preceding activities.
*/

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

```
/*
      Earliest Finish Time :
      The earliest finish time for each activity is calculated by
      adding the relevant activity duration to its earliest start time.
*/

Earliest_Finish_Time(A : Es + D):-
      Earliest_Start_Time(A : Es),
      Duration(A : D).

/*
      Latest Finish Time :
      The latest finish time for each activity is found by
      taking the minimum of the latest finish time - duration
      for each of its successing activities.
*/

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

/*
      Latest Start Time :
      The latest start time for each activity is calculated by
      subtracting the relevant activity duration from its latest
      finish time.
*/

Latest_Start_Time(A : Lf - D):-
      Latest_Finish_Time(A : Lf),
      Duration(A : D).

/*
      Total Float :
      The total float for each activity is its earliest finish time
      subtracted from its latest finish time.
*/

Total_Float(A : Lf - Ef):-
      Latest_Finish_Time(A : Lf),
      Earliest_Finish_Time(A : Ef).


/*
      Resource & Cost Analysis
*/

resource_analysis:-
```

```
      time_constraints,
      resource_analysis1(0).

resource_analysis1(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,
      resource_analysis_summary(Time, A, B, C),
      (Active(end, Time : on),
            nl, write("Project Duration = "), print("%3.0Lf", Time), nl,
            Total_Cost(TC),
            write("Total Direct Cost(NPV) = "), print("%7.0Lf", TC),nl,
            dump([TC]) ;
            resource_analysis1(Time + 1)
      ),!.

resource_analysis_summary(Time, A, B, C):-
      write("Week "), print("%3.0Lf", Time), write(" : "),nl,
      bagof(X, (Diagram(X, _, _), Active(X, Time : on)), As),
      tab, write("Active Processes = "), write(As), nl,
      tab, write("Engineers  = "), print("%3.0Lf", A), nl,
      tab, write("Power Units = "), print("%3.0Lf", B), nl,
      tab, write("Draftsmen  = "), print("%3.0Lf", C), nl, nl.

Finish(Time):- Start_Time(end : St), St <= Time.

Activities(: As):- bagof(A, Diagram(A, _, _), As).

/* Time Constraints */

Start_Time(A : St).

time_constraints:-
      Activities(: As),
      time_constraints1(As),!.

time_constraints1([]).
time_constraints1([A|R]):-
      Diagram(A, N, _),
      bagof(B, Diagram(B, _, N), Bs),
      time_constraints2(A, Bs),
      time_constraints1(R).

time_constraints2(A, []):-
      Start_Time(A : St),
      St >= 0.
time_constraints2(A, [B|R]):-
      Start_Time(A : St),
      Start_Time(B : St1),
      Duration(B : D1),
      St1 + D1 <= St,
```

```
      time_constraints2(A, R).

/* Scheduling */

Active(A, -1 : off):-!.
Active(A, Time : X):- Time >= 0.

scheduling(Time):-
      Activities(: As),
      scheduling1(Time, As).

scheduling1(Time, []).
scheduling1(Time, [A|R]):-
      Start_Time(A : St),
      Duration(A : D),
      scheduling2(A, Time, St, D),
      scheduling1(Time, R).

scheduling2(A, Time, Time, D):-
      Active(A, Time - 1 : off),
      Active(A, Time : on).
scheduling2(A, Time, St, D):-
      Active(A, Time - 1 : on),
      Time < St + D,
      Active(A, Time : on).
scheduling2(A, Time, _, _):-
      Active(A, Time : off).

/* Resources */

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).

/* Cost Analysis */

Net_Present_Value(Rate, Weeks, V, V / (1 + R)):-
      R = Rate * Weeks * 7 / 365.

Total_Cost(TC):-
      discount_rate(: Rate),
      sum(V1, (
            Diagram(X, _, _),
            Start_Time(X : St),
            Cost(X : V),
            Net_Present_Value(Rate, St, V, V1)
            ),
      TC).

/*
-----------------------------------------------------------------------
      Car Manufacture Model
```

```
------------------------------------------------------------------------
*/

/* Discount Rate */

discount_rate(: 0.12).

/* Diagram */

Diagram(start, n0, n1).
Diagram('Design_Engine', n1, n2).
Diagram('Design_Body', n1, n4).
Diagram('Order_Body_Metal', n1, n3).
Diagram('Deliver_Body_Metal', n3, n4).
Diagram(dummy, n2, n4).
Diagram('Manufacture_Body', n4, n5).
Diagram('Manufacture_and_Test_Engine', n2, n5).
Diagram('Assemble_Car', n5, n6).
Diagram(end, n6, n7).

/* Duration */

Duration(start : 0).
Duration('Design_Engine' : 5).
Duration('Design_Body' : 4).
Duration('Order_Body_Metal' : 1).
Duration('Deliver_Body_Metal' : 2).
Duration(dummy : 0).
Duration('Manufacture_Body' : 4).
Duration('Manufacture_and_Test_Engine' : 3).
Duration('Assemble_Car' : 2).
Duration(end : 0).

/* Resources */

Resources(start, engineers : 0).
Resources('Design_Engine', engineers : 1).
Resources('Design_Body', engineers : 1).
Resources('Order_Body_Metal', engineers : 0).
Resources('Deliver_Body_Metal', engineers : 0).
Resources(dummy, engineers : 0).
Resources('Manufacture_Body', engineers : 2).
Resources('Manufacture_and_Test_Engine', engineers : 3).
Resources('Assemble_Car', engineers : 1).
Resources(end, engineers : 0).

Resources(start, power_units : 0).
Resources('Design_Engine', power_units : 0).
Resources('Design_Body', power_units : 0).
Resources('Order_Body_Metal', power_units : 0).
Resources('Deliver_Body_Metal', power_units : 5).
Resources(dummy, power_units : 0).
Resources('Manufacture_Body', power_units : 15).
Resources('Manufacture_and_Test_Engine', power_units : 20).
Resources('Assemble_Car', power_units : 15).
Resources(end, power_units : 0).

Resources(start, draftsmen : 0).
```

```
Resources('Design_Engine', draftsmen : 2).
Resources('Design_Body', draftsmen : 2).
Resources('Order_Body_Metal', draftsmen : 0).
Resources('Deliver_Body_Metal', draftsmen : 0).
Resources(dummy, draftsmen : 0).
Resources('Manufacture_Body', draftsmen : 0).
Resources('Manufacture_and_Test_Engine', draftsmen : 0).
Resources('Assemble_Car', draftsmen : 0).
Resources(end, draftsmen : 0).

/* Resource Constraints */

Resource_Limit(engineers : 4).
Resource_Limit(power_units : 25).
Resource_Limit(draftsmen : 2).

/* Cost */

Cost(start : 0).
Cost('Design_Engine' : 10000).
Cost('Design_Body' : 35000).
Cost('Order_Body_Metal' : 2500).
Cost('Deliver_Body_Metal' : 2000).
Cost(dummy : 0).
Cost('Manufacture_Body' : 35000).
Cost('Manufacture_and_Test_Engine' : 25000).
Cost('Assemble_Car' : 30000).
Cost(end : 0).
```

Appendix III

```
/*

      A Model for Setupping a computer-bases corporate information system

            Vasant Dhar and Albert Croker,
            Knowledge-Bases Decision Support in Business:
            Issues and a Solution,
            IEEE EXPERT, Spring 1988, pp53-62

*/


/*
      Hardware Specifications
      Name : Cost(K), MIPS, Memory(MB), Annual maintenance
*/

Hardware('DEC').
Hardware('IBM').
Hardware('CDC').
Hardware('ATT').

HW_Spec('DEC' : 300000, 5, 8, low   ).
HW_Spec('IBM' : 500000, 5, 8, high  ).
HW_Spec('CDC' : 350000, 5, 8, high  ).
HW_Spec('ATT' : 100000, 2, 4, medium).


/*
      Air-conditioning Units
      Name : Cost(K), Power(KW), Space(K sqft)
*/

Air_Conditioning('Borg').
Air_Conditioning('Westinghouse').
Air_Conditioning('GE').

AC_Spec('Borg'        : 20000, 200, 2).
AC_Spec('Westinghouse' : 35000, 500, 3).
AC_Spec('GE'          : 30000, 300, 3).


/*
      Operating Systems
      Name : Cost(K), Memory required(MB)
*/

Operating_System('UNIX').
Operating_System('VMS').
Operating_System('CMS').
Operating_System('TOPS').

OS_Spec('UNIX' : 10000, 1.5).
OS_Spec('VMS'  : 20000, 2.0).
OS_Spec('CMS'  : 40000, 2.0).
```

```
OS_Spec('TOPS' :  5000, 1.0).


/*
      Database Management Systems
      Name : Cost(K), Memory required(MB)
*/

Database('INGRES').
Database('IMS').
Database('RIM').
Database('DMS').

DB_Spec('INGRES' :  7000, 200,  200).
DB_Spec('IMS'    : 15000, 350, 1000).
DB_Spec('RIM'    :  4000, 100,  150).
DB_Spec('DMS'    : 10000, 150,  200).


/* selection */

selection(: HW, AC, OS, DB, N):-
      Hardware(HW),
      Air_Conditioning(AC),
      Operating_System(OS),
      Database(DB),
      N >= 1.


/* Constraints */

constraints :- quantative_constraints, qualitative_constraints.

quantative_constraints :-

      selection(: HW, AC, OS, DB, N),

      HW_Spec(HW : HWcost, HWmips, HWmem, HWmaint),
      AC_Spec(AC : ACcost, ACpower, ACspace),
      OS_Spec(OS : OScost, OSmem),
      DB_Spec(DB : DBcost, DBmem, DBrec),

      /* at least 6M bytes must be available
      apart from memory requirements of the operating system
      and the database system. */

      HWmem - OSmem - DBmem/1024 >= 6.0,

      /* the total cost of the various information system components
      should not exceed $375,000 */

      HWcost + ACcost + OScost + N * DBcost <= 375000.

qualitative_constraints :-

      selection(: HW, AC, OS, DB, N),

      HW_Spec(HW : HWcost, HWmips, HWmem, HWmaint),
      AC_Spec(AC : ACcost, ACpower, ACspace),
      OS_Spec(OS : OScost, OSmem),
```

```
      DB_Spec(DB : DBcost, DBmem, DBrec),

      /* Rule 1 :
            if IBM is selected as the hardware
            then the OS must be CMS
      */

      IF( EQ(HW, 'IBM'), EQ(OS, 'CMS') ),

      /* Rule 2 :
            if IBM is selected as the hardware
            then the AirConditioning power must be greater than 400 KW
      */

      IF( EQ(HW, 'IBM'), (ACpower > 400) ),

      /* Rule 3 :
            if the Hardware is DEC and the OS is UNIX
            then INGRES must be selected as the DBMS.
      */

      IF( ( EQ(HW, 'DEC'), EQ(OS, 'UNIX') ), EQ(DB, 'INGRES') ),

      /* Rule 4 :
            if IMS is selected as the OS
            then the hardware must be IBM.
      */

      IF( EQ(OS, 'IMS'), EQ(HW, 'IBM') ),

      /* Rule 5 :
            if INGRES is selected as the DBMS
            then the hardware must not be CDS.
      */

      IF( EQ(DB, 'INGRES'), NEQ(HW, 'CDC') ),

      /* Rule 6 :
            VMS operating system should not be installed
            on hardware that is less than 5 MIPS
      */

      IF( (HWmips < 5), NEQ(OS, 'VMS') ).


/* Equations */

TotalCost(: HWcost + ACcost + OScost + N * DBcost) :-

      selection(: HW, AC, OS, DB, N),

      HW_Spec(HW : HWcost, HWmips, HWmem, HWmaint),
      AC_Spec(AC : ACcost, ACpower, ACspace),
      OS_Spec(OS : OScost, OSmem),
      DB_Spec(DB : DBcost, DBmem, DBrec).

SpaceRequired(: ACspace * 1000 + 1000):-
      selection(: HW, AC, OS, DB, N),
      AC_Spec(AC : ACcost, ACpower, ACspace).


/* Goal */

Feasible(HW, AC, OS, DB, N, TotCost, HWmips):-
```

```
      selection(: HW, AC, OS, DB, N),
      constraints,
      TotalCost(:TotCost),
      HW_Spec(HW : HWcost, HWmips, HWmem, HWmaint).

write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space):-
      write("Ans :"),nl,
      tab,write("Hardware Maker    = "),write(HW),
            write(" ("),print("%2.0Lf", HWmips),write(" MIPS)"),nl,
      tab,write("Air-Conditioner   = "),write(AC),nl,
      tab,write("Operating System  = "),write(OS),nl,
      tab,write("Database Software = "),write(DB),
            write(" * "), print("%2.0Lf", N),nl,
      tab,write("-----------------------------------"),nl,
      tab,write("Total Cost       = "),print("%6.0Lf", TotCost),nl,
      tab,write("Space           = "),print("%6.0Lf", Space),nl,nl,
      dump([N, TotCost]).

/* mininize Total Cost */
go1:-
      min(TotCost, Feasible(HW, AC, OS, DB, N, TotCost, HWmips)),
      SpaceRequired(: Space),
      write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).

/* maximizing MIPS */
go2:-
      max(HWmips, Feasible(HW, AC, OS, DB, N, TotCost, HWmips)),
      SpaceRequired(: Space),
      write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).

/* combination of two objectives */
go3:-
      min(TotCost,
            max(HWmips, Feasible(HW, AC, OS, DB, N, TotCost, HWmips))
      ),
      SpaceRequired(: Space),
      write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).

go4:-
      max(HWmips,
            min(TotCost, Feasible(HW, AC, OS, DB, N, TotCost, HWmips))
      ),
      SpaceRequired(: Space),
      write_ans(HW, AC, OS, DB, N, TotCost, HWmips, Space).

/* Utility */

EQ(X, Y) :- unify(X, Y).

NEQ(X, Y) :- unify(X, Y),!,fail.
NEQ(_, _).

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

RANGE(V, X, L, U):- min(V, X, L), max(V, X, U).
```