TR-476

A Formalization of Modeling on Relational Data — Utilization of Constraint Logic Programming for Problem Solving —

by
K. Hiraishi

May, 1989

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Abstract

We propose a mathematical formalization of modeling which is based on the relational data model. A model is described as a set of functions, each of which represents the functional dependency in each relation. A wide variety of models can be described with this formalization. Moreover, operations for models are introduced. These operations enable flexible approaches of modeling. Capabilities of this formalization are demonstrated by a model description language MODEL/R and its interpretation system on a constraint logic programming language CLP(R).

1. Introduction

Modeling is an approach to problem solving and decision making. Various kinds of models are proposed in many fields, especially in operations research and management science. Computer-based *decision support systems*(DSS) available in markets usually store these models as program packages, and provides functions to call them and display/print their results in suitable formats. However, the task to choose an appropriate model and to write down the problem in the form specified for the model is left to users.

Model Management Systems(MMS) have been proposed to support the process of managing models, and to provide functions of creation, manipulation of and access to models. Several frameworks for model representation have been proposed to implement MMS, such as *predicate calculus*[Bonczek 81], *production rules*[Holsapple 86], *semantic networks*[Elam 80], *frames*[Dolk 84] and *relational model*[Blanning 86]. These approaches pay more attention to managing models than describing models.

Structured Modeling[Geoffrion 87] emphasizes the process of expressing user's problems, and proposes a framework of describing models in a unified manner. Structured Modeling is based on the assumption that a model is an essential conceptual unit. This paper takes the same standpoint as Structured Modeling, and proposes a mathematical formalization of modeling which is based on the relational data model. A wide variety of models can be described with this formalization.

In Section 2, the concept of modeling is described. A model is defined as a mapping from a data structure to data occurrences in the real world. The relational data model is used for the data structure. Relations are assumed to be in third normal form and a model is described as a set of functions, where each function represents the functional dependency of each relation. In Section 3, modeling is formalized on relational data. In Section 4, problem solving through modeling is discussed. In Section 5, operations for models are defined. These operations enable various kinds of modeling approaches. Moreover, we specify the model description language MODEL/R and implement its interpretation system on a constraint logic programming language CLP(R). Brief instruction of MODEL/R and some examples are shown in Section 6.

2. Data Structures and Models

In the data base theory, data models are proposed for formalizing and expressing data in the real world. In other words, the data model theory aims to define a data structure which reflects semantics in the real world. A data base is considered to be *an extension* (or *instantaneous value*) of the data structure. In science and engineering, modeling aims to construct a mechanism which explains how data observed in experiments are generated. Therefore, we can consider a model as *an intension* of the data structure. Fig.1 shows the relation among a data structure, data occurrences(data base) and a model.

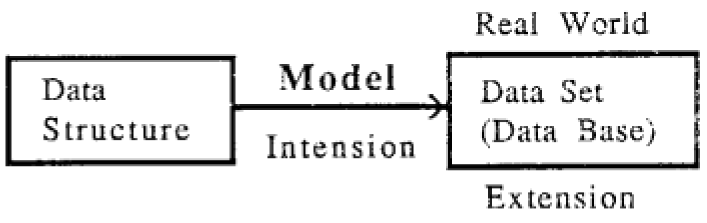

Fig.1 Concept of Modeling

We use the relational data model as a data structure. *A relation* R is defined as a subset of a cartesian product set $R \subset A_1 \times \cdots \times A_n$, where $A_i (i=1,\cdots,n)$ is called *an attribute*. We write $R[A_1,\cdots,A_n]$ to indicate attributes of a relation R. Let X and Y be any collections of attributes. Then Y is called *functionally dependent* on X if the value of Y is uniquely determined by the value of X. Functional dependency is not a temporary property. It is always satisfied as a semantical constraint in the real world.

A relation is decomposed into simple relations by using functional dependency. *The third normal form(Boyce-Codd normal form)* [Codd 74] is the most popular decomposition. A relation R is in third normal form if the following holds for any attribute collection X:

"If any attribute not in X is functionally dependent on X, then all attributes in R are functionally dependent on X."

A relation takes various values in various situations. Considering a relation STUDENT [Number, Name, Address], every school has its own value for STUDENT. Moreover, even in the same school, the value differs every year. In this case, *school* and *year* represent situations for STUDENT to take different values. We use the word *index* for such a description that determines an extension of a relation, e.g., a set of school names is an index for the relation STUDENT. An intension of a relation $R[A_1,\cdots,A_n]$ is described as a function from some indices to the power set of $A_1\times\cdots\times A_n$.

A relation R can represent the situations specified by an index i by adding i as a key attribute. For example, the relation STUDENT[School, Number, Name, Address] includes information about students in each school. From the meaning of an index, the attribute for an index is not functionally dependent on other attributes.

Let R be a relation in third normal form, X be a set of key attributes of R, and Y be the set of attributes not in X. Since the value of Y is uniquely determined for a given value of X, X is considered to be indices for Y. Moreover, R can be described as an intensional form

$$R=\{<x,y> \mid x\in R_x \wedge y=f(x)\},$$

where R_x is the projection of R to X, R_y is the projection of R to Y, and $f:R_x\rightarrow R_y$ is a function. A model for R is expressed by the function f.

3. Formalization of Modeling

In this section, we formalize modeling on relational data mentioned above.

3.1 Schema and Model

We first define *union of functions* for preparation. Let $f_1:X_1\rightarrow Y_1$ and $f_2:X_2\rightarrow Y_2$ be functions such that $f_1(x)=f_2(x)$ for any $x\in X_1\cap X_2$. Then the function $f_1\cup f_2:X_1\cup X_2\rightarrow Y_1\cup Y_2$ is defined by

i) $f_1\cup f_2(x)=f_1(x)$ for any $x\in X_1$;

ii) $f_1\cup f_2(x)=f_2(x)$ for any $x\in X_2$.

We define the schema description language S which specifies data structures. S consists of the following symbols:

i) the set of *primitive types* $P:=\{p_1,p_2,\cdots\}$;
ii) the set of *derived types* $T:=\{t_1,t_2,\cdots\}$;
iii) the set of *relation names* $N:=\{r^1_{<0,1>},r^2_{<0,1>},\cdots, r^1_{<0,2>},r^2_{<0,2>},\cdots, r^1_{<n,m>},r^2_{<n,m>},\cdots\}$;
iv) the set of *constraints* $C:=\{c_1,c_2,\ldots\}$;
v) *connective symbols*: $|$, $[$, $]$, $=$, $:$, $,$.

S has two types of formulas:
i) "$t=t'|c$" is called *a type formula* for t if $t\in T$, $t'\in P\cup T$ and $c\in C$;
ii) "$r^k_{<n,m>}=[a_1,\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$" is called *a relation formula* for $r^k_{<n,m>}$ if $r^k_{<n,m>}\in N$ and $a_i\in P\cup T(i=1,\cdots,n+m)$. In the case $n=0$ or $m=0$, we write $r^k_{<n,m>}=[:a_{n+1},\cdots,a_{n+m}]$ or $r^k_{<n,m>}=[a_1,\cdots,a_n:]$, respectively. $a_1,\cdots,a_n$ are called *key attributes* of $r^k_{<n,m>}$ and $a_{n+1},\cdots,a_{n+m}$ are called *non-key attributes* of $r^k_{<n,m>}$.

A set of formulas G is called a schema if the following holds:
i) G has at least one relation formula;
ii) G has exactly one type formula for every $t\in T$ appeared in G.

Let T_G denote the set of symbols in T which appear in a schema G, and similarly for N_G. $B=<U,\pi>$ is called *a primitive model* if U is a nonempty set and $\pi:P\cup C\rightarrow 2^U$ is a function, i.e., $\pi(x)\subset U$ for each $x\in P\cup C$. A primitive model specifies semantics for primitive types and constraints. For a schema G, let D be a non-empty set such that $D\subset U$, and μ be a function such that
i) the domain of μ is $T_G\cup N_G$;
ii) $\mu(t)\subset D$ for each $t\in T_G$;
iii) for each $r^k_{<n,m>}\in N_G$, $\mu(r^k_{<n,m>})$ is a partial function from D^n to D^m,
where D^k denotes $D\times\cdots\times D$(k times) for a nonnegative integer k. Let μ^+ denote $\mu\cup\pi$. $M=<D,\mu>$ is called a model for G if the following holds:
i) $\mu(t)\subset\mu^+(t')\cap\pi(c)$ for each type formula $t=t'|c$ in G;
ii) $\mu(r)(x)\subset\mu^+(a_{n+1})\times\cdots\times\mu^+(a_{n+m})$ for each relation formula $r=[a_1,\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ in G and each $x\in dom_M(r)$, where $dom_M(r)$ is the domain of $\mu(r)$ in $\mu^+(a_1)\times\cdots\times\mu^+(a_n)$.

Let $M=<D,\mu>$ be a model for a schema G and $r=[a_1,\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ be a relation formula in G. Then the extension of r in M is defined by

$$\begin{aligned} ext_M(r)=\{&<x_1,\cdots,x_n,y_1,\cdots,y_m> \mid <x_1,\cdots,x_n>\in dom_M(r) \\ &\wedge <y_1,\cdots,y_m>=\mu(r)(<x_1,\cdots,x_n>)\}, \end{aligned}$$

where ext_M is a function which maps each relation name in G to its extension in M. We call ext_M *the solution* of M. Let $range_M(r)$ denote the set $\mu^+(a_1)\times\cdots\times\mu^+(a_{n+m})$. Clearly $ext_M(r)\subset range_M(r)$.

3.2 Example of Modeling

An example is used for illustrating the modeling process defined above. Feedmix model[Geoffrion 87] below describes how to blend nutrients with animal feed. There are two kinds of nutrients, protein and calcium, and there are two kinds of materials, standard and additive. The minimum daily requirement is given for each nutrient. Unit cost and quantity are given for each material. Analysis indicates quantity of each nutrient blended with each material per unit.

We first define a schema:

pounds	= real $\mid \geqq 0$;
dollars	= real $\mid \geqq 0$;
nutrient	= chr $\mid$ true;
material	= chr $\mid$ true;
MIN	= [nutrient : pounds];
UCOST	= [material : dollars];
Q	= [material : pounds];
ANALYSIS	= [nutrient, material : pounds];
NLEVEL	= [nutrient : pounds];
T:NLEVEL	= [nutrient : boolean];
TOTCOST	= [: dollars],

where 'chr', 'real' and 'boolean' are basic types, 'true' and '$\geqq 0$' are constraints such that $\pi(\text{true})=U$ and $\pi(\geqq 0)$ is the set of nonnegative numbers in U. NLEVEL indicates the quantity of each nutrients in feed. T:NLEVEL is true if the value of NLEVEL is greater than the value of MIN for each nutrient, otherwise false. TOTCOST is the total cost used for feed. No key attributes are assigned to TOTCOST. This means that TOTCOST can take only one value in this schema.

A model $M=<D,\mu>$ for the schema above is defined as follows:

i) $D=\mathbf{R}\cup\mathbf{W}$, where **R** is the set of real numbers and **W** is the set of words;

ii) μ is defined by

pounds	$\rightarrow \{x \mid x \in R \wedge x \geqq 0\}$;
dollars	$\rightarrow \{x \mid x \in R \wedge x \geqq 0\}$;
nutrient	$\rightarrow$ {protein, calcium};
material	$\rightarrow$ {standard, additive};
MIN	$\rightarrow$ {protein$\rightarrow$6, calcium$\rightarrow$4};
UCOST	$\rightarrow$ {standard$\rightarrow$1.2, additive$\rightarrow$3};
Q	$\rightarrow$ {standard$\rightarrow$2, additive$\rightarrow$0.5};
ANALYSIS	$\rightarrow$ {<protein, standard>$\rightarrow$4,
	<protein, additive>$\rightarrow$14,

$\langle$calcium, standard$\rangle \rightarrow 2$,
$\langle$calcium, additive$\rangle \rightarrow 1\}$;

TOTCOST $\rightarrow SUM(\{\mu(\mathrm{UCOST})(m)*\mu(\mathrm{Q})(m) \mid m\in\mu(\mathrm{material})\})$;

NLEVEL $\rightarrow [n\in\mu(\mathrm{nutrient})\rightarrow$
$SUM(\{\mu(\mathrm{ANALYSIS})(n,m)*\mu(\mathrm{Q})(m) \mid m\in\mu(\mathrm{material})\})]$;

T:NLEVEL $\rightarrow \left[n\in\mu(\mathrm{nutrient})\rightarrow \begin{cases}\text{true if } \mu(\mathrm{NLEVEL})(n)\geqq\mu(\mathrm{MIN})(n); \\ \text{false otherwise}\end{cases}\right]$,

where $x\rightarrow y$ denotes the mapping from x to y, and $SUM(R)$ denotes the function which returns the summation of each element in R.

3.3 Dependency among Relations

In Feedmix model, μ(NLEVEL) refers to μ(ANALYSIS) and μ(Q). Hence. values of ANALYSIS and Q determine the value of NLEVEL. We use the word *dependency* for such a relation. In this case, NLEVEL is said to be *dependent* on ANALYSIS and Q, and we write NLEVEL>ANALYSIS and NLEVEL>Q to indicate the dependency. Fig.2 shows dependency among relations in Feedmix model. An upper relation is dependent on lower relations connected with lines.

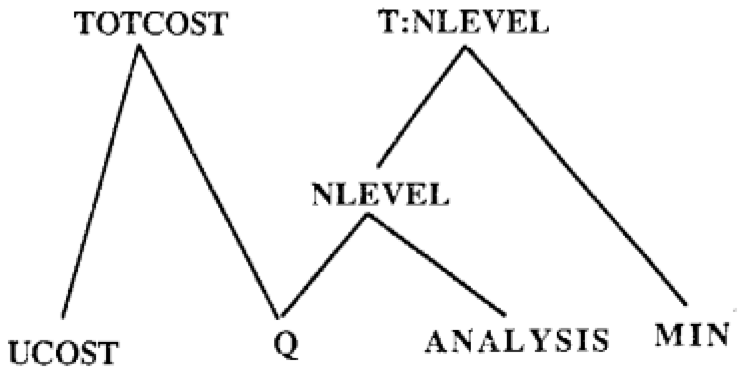

Fig.2 Dependency among Relations in Feedmix Model

We can explicitly write the dependency with μ(T:NLEVEL) by the following form:

$$\mu(\text{T:NLEVEL}) : \mu(\text{nutrient}) \times 2^{range_M(\text{NLEVEL})} \times 2^{range_M(\text{MIN})} \rightarrow \pi(\text{boolean}),$$

which is defined by

$$\mu(\text{T:NLEVEL})(n, ext_M(\text{NLEVEL}), ext_M(\text{MIN})) = \begin{cases} \text{true if } x \geqq y \\ \text{false otherwise} \end{cases}$$

where $n \in \mu(\text{nutrient}) \wedge \langle n,x \rangle \in ext_M(\text{NLEVEL}) \wedge \langle n,y \rangle \in ext_M(\text{MIN})$.

Similarly,

$$\mu(\text{TOTCOST})(ext_M(\text{UCOST}), ext_M(\text{Q})) = SUM(\{x^*y \mid \exists m \in \mu(\text{material}) : \langle m,x \rangle \in ext_M(\text{UCOST}) \wedge \langle m,y \rangle \in ext_M(\text{Q})\}),$$

$$\mu(\text{NLEVEL})(n, ext_M(\text{ANALYSIS}), ext_M(\text{Q})) = SUM(\{x^*y \mid \exists m \in \mu(\text{material}) : \langle n,m,x \rangle \in ext_M(\text{ANALYSIS}) \wedge \langle m,y \rangle \in ext_M(\text{Q})\}),$$

where $n \in \mu(\text{nutrient})$.

Let $r=[a_1,\dots,a_n:a_{n+1},\dots,a_{n+m}]$ be a relation formula in a schema G. Considering the dependency of r, $\mu(r)$ can be described in the following form:

$$\mu(r) : \mu^+(a_1) \times \cdots \times \mu^+(a_n) \times 2^{range_M(r_1)} \times 2^{range_M(r_k)} \rightarrow \mu^+(a_{n+1}) \times \cdots \times \mu^+(a_{n+m}),$$

where r_i is a relation name in G such that $r > r_i (i=1,\cdots,k)$. By this definition, $ext_M(r)$ is expressed by

$$ext_M(r) = \{\langle x_1,\cdots,x_n,y_1,\cdots,y_m \rangle \mid \langle x_1,\cdots,x_n \rangle \in dom_M(r) \wedge \langle x_1,\cdots,x_n,y_1,\cdots,y_m \rangle = \mu(x_1,\cdots,x_n, ext_M(r_1),\cdots,ext_M(r_k))\}.$$

Dependency among relations satisfies transitivity, i.e., $r_i > r_j$ and $r_j > r_k$ imply $r_i > r_k$. A model M for a schema G is called *acyclic* if $r_j > r_i$ does not hold for any two relations r_i and r_j in M such that $r_i > r_j$. If a model M for a schema G is acyclic, then we can obtain the solution of M by the following algorithm.

[Algorithm]

1. Let E be the set of relations in G.
2. Select relations which are minimal in E for the dependency '>', where r is called minimal in E if there does not exist $x \in E$ such that $r > x$. Compute $ext_M(r)$ for each minimal element r and go to 3.
3. Let $E := E - \{r\}$. If $E = \phi$, then the solution is obtained. Otherwise, go to 2.

Since Feedmix model is acyclic, the solution is obtained by the following process:

In the first recursion,

1. E={MIN, UCOST, Q, ANALYSIS, TOTCOST, NLEVEL, T:NLEVEL}.
2. MIN, UCOST, Q and ANALYSIS are minimal in E, and

   ext_M(MIN)={<protein, 16>, <calcium, 4>},

   ext_M(UCOST)={<standard, 1.2>, <additive, 3>},

   ext_M(Q)={<standard, 2>, <additive, 0.5>},

   ext_M(ANALYSIS)={<protein, standard, 4>, <protein, additive, 14>, <calcium, standard, 2>, <calcium, additive, 1>}

are immediately obtained.

3. E={TOTCOST, NLEVEL, T:NLEVEL}.

In the second recursion,

2. TOTCOST and NLEVEL are minimal, and

   ext_M(TOTCOST)={< 3.9 >},

   ext_M(NLEVEL)={<protein, 15>, <calcium, 4.5>}.

3. E={T:NLEVEL}.

In the third recursion,

2. T:NLEVEL is minimal and

   ext_M(T:NLEVEL)={<protein, false>, <calcium, true>}.

3. $E = \phi$.

If a model is cyclic, then there exists a relation r such that $r > r_1 > r_2 > \cdots > r$ and we must use an appropriate *constraint solver* to find the solution. For example, if each function in a model is composed of linear equations or inequalities, then the solution can be computed by a linear constraint solver such as *the simplex method.*

4. Problem Solving through Modeling

One of the purposes of building Feedmix model is to find values of quantity which minimize the total cost, satisfying the minimal daily requirement. The notion of optimization is not included in Feedmix model. Optimization methods are applied to Feedmix model for the purpose "minimizing the total cost". There can exist various purposes which require different solving techniques. In this section, we discuss methods for problem solving through modeling.

4.1 Finding Solutions

In Feedmix model, one possible way to find optimum value is finding solutions for various initial values of quantity. *The what-if analysis*, which is a method to find effects of changing some parts of a model, and *the sensitivity analysis*, which is a method to find effects of changing values step by step, are conducted for evaluating effects of uncertainty. In addition, *the direct search method*, which is a method for unconstrained non-linear programming, can be used for searching solutions.

4.2 Extracting Formulas for Solvers

Feedmix model has two decision variables, which are X_s:quantity for standard feed and X_a:quantity for additive feed. We can easily extract formulas expressed with X_s and X_a from Feedmix model.

Minimize

$$1.2X_s+4X_a$$

such that

$$4X_s+14X_a \geqq 16$$
$$2X_s+\ X_a \geqq 4$$

This optimization problem can be solved by the linear programming(LP) method. If an LP package requires a matrix form, these formulas must be converted into matrices. If extracted formulas contain non-linear terms, then an appropriate non-linear programming package would be required. Since most solvers require formulas(or matrix forms), the first step to use solvers is extracting formulas out of the model.

4.3 Constraint Programming

Constraint programming languages enable us to use constraint solvers more easily. A constraint programming language is a computer language that contains a solver in its processing system[Leler 88]. In ordinary computer languages, such as C or Pascal, a program is an algorithm to be executed step by step. In constraint languages, a program is a set of relations between a set of objects, and it is the job of the constraint satisfaction system to find a solution that satisfies these relations. Describing a model on an appropriate constraint language, the solution is automatically computed by the constraint-satisfaction system. Constraint logic programming languages, which realize a constraint satisfaction mechanism on the logic programming framework, is suitable for representing relational data with numerical constraints. In Section 6, we show some examples of problem solving by using a constraint logic programming language.

5. Operations for Models

In this section, operations for models are defined. These operations enable us to edit models without going into their internal descriptions, i.e., we can treat a model as a unit for editing.

5.1 Definitions of Operations

(1) Join

Join is an operation to combine some models together. Let $M_1=<D_1,\mu_1>$ and $M_2=<D_2,\mu_2>$ be models for schema G_1 and G_2, respectively. $\mu_1 \cup \mu_2$ can be defined if $\mu_1(x)=\mu_2(x)$ holds for any $x \in (T_{G_1} \cap T_{G_2}) \cup (N_{G_1} \cap N_{G_2})$. Then $M=<D_1 \cup D_2, \mu_1 \cup \mu_2>$ is a model for $G_1 \cup G_2$. The operation to make M from M_1 and M_2 is called join.

(2) Projection

Projection is an operation to make a model for projected relations. Let M be a model for a schema G. We define the projection of M to a key attribute $t \in T_G$ and $v \in \mu(t)$. we make a projected schema G_P by i) replacing each relation formula $r=[a_1,\cdots,a_{i-1},t,a_{i+1},\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ in G, which has t as a key attribute, with $r_P=[a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n:a_{n+1},\ldots,a_{n+m}]$, ii) excluding unnecessary type definitions. Let $\tilde{N}_G$ be the set of replaced relation names in N_G. A projection of M to t and v is $M_P=\langle D,\mu_P\rangle$ such that

i) $\mu_P(x)=\mu(x)$ holds for each $x \in T_{G_P} \cup N_G \backslash \tilde{N}_G$;

ii) $\mu_P(r_P)(\langle x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n\rangle)=\mu(r)(x)$ holds for each $r \in \tilde{N}_G$ and each $x=\langle x_1,\cdots,x_{i-1},v,x_{i+1},\cdots,x_n\rangle \in dom_M(r)$, where $r=[a_1,\cdots,a_{i-1},t,a_{i+1},\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ is in G and its replacement in G_P is $r_P=[a_1,\cdots,a_{i-1},a_{i+1},\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$.

A projection of Feedmix model for index attribute 'material' and its value 'standard feed' is shown as follows.

Projected Schema G_P :

```
pounds      = real | ≧0;
dollars     = real | ≧0;
nutrient    = chr | true;
MIN         = [nutrient : pounds];
UCOST       = [: dollars];
Q           = [: pounds];
ANALYSIS    = [nutrient: pounds];
NLEVEL      = [nutrient : pounds];
T:NLEVEL    = [nutrient : boolean];
TOTCOST     = [: dollars],
```

Projected Model $M_P=\langle D,\mu_P\rangle$:

pounds	$\rightarrow \{x \mid x \in R \wedge x \geqq 0\}$;
dollars	$\rightarrow \{x \mid x \in R \wedge x \geqq 0\}$;
nutrient	$\rightarrow$ {protein, calcium};
MIN	$\rightarrow$ {protein$\rightarrow$16, calcium$\rightarrow$4};
UCOST	$\rightarrow 1.2$;
Q	$\rightarrow 2$;
ANALYSIS	$\rightarrow$ {protein$\rightarrow$4, calcium$\rightarrow$2};
TOTCOST	$\rightarrow \mu(\text{UCOST})*\mu(\text{Q})$;
NLEVEL	$\rightarrow [n \in \mu(\text{nutrient}) \rightarrow \mu(\text{ANALYSIS})(n)*\mu(\text{Q})]$;
T:NLEVEL	$\rightarrow \left[n \in \mu(\text{nutrient}) \rightarrow \begin{cases} \text{true if } \mu(\text{NLEVEL})(n) \geqq \mu(\text{MIN})(n); \\ \text{false otherwise} \end{cases}\right]$,

(3) Enlargement

Enlargement is an operation to add an index as a key attribute to each relation. Let $M=<D,\mu>$ be a model for a schema G. We define the enlargement of M with an index I and a value v. Let $\tilde{N}_G$ be the set of relation names in N_G which have I as an index. We make an enlarged schema G_E from G by i) adding type formulas for I, ii) replacing each relation formula $r=[a_1,\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ for $r\in\tilde{N}_G$ with $r_E=[I,a_1,\cdots,a_n:a_{n+1},\cdots, a_{n+m}]$.

An enlarged model with I and v is $M_E=<D,\mu_E>$ such that

i) $\mu_E(x)=\mu(x)$ holds for each $x\in T_G \cup N_G\backslash\tilde{N}_G$,

ii) $\mu_E(r_E)(<v,x_1,\cdots,x_n>)=\mu(r)(x)$ holds for each $r\in\tilde{N}_G$ and each $x=<x_1,\cdots,x_n>\in$ $dom_M(r)$, where $r=[a_1,\cdots,a_n:a_{n+1},\cdots,a_{n+m}]$ is in G and its replacement in G_E is $r_E=[I,a_1,\cdots,a_n:a_{n+1},\cdots, a_{n+m}]$.

Assume that Feedmix model is described in consideration of pigs. Then we can make an enlarged model by adding a key attribute 'animal' and its value 'pig' to relations in Feedmix model. UCOST and ANALYSIS does not change because they depend only on materials.

Enlarged Schema G_E :

pounds	$= \text{real} \mid \geqq 0;$
dollars	$= \text{real} \mid \geqq 0;$
animal	$= \text{chr} \mid \text{true};$
nutrient	$= \text{chr} \mid \text{true};$
material	$= \text{chr} \mid \text{true};$
MIN	= [animal, nutrient : pounds];
UCOST	= [material : dollars];
Q	= [animal, material : pounds];
ANALYSIS	= [nutrient, material : pounds];
NLEVEL	= [animal, nutrient : pounds];
T:NLEVEL	= [animal, nutrient : boolean];
TOTCOST	= [animal : dollars].

Enlarged Model $M_E=<D,\mu_E>$:

pounds	$\rightarrow\{x \mid x\in R \wedge x\geqq 0\};$
dollars	$\rightarrow\{x \mid x\in R \wedge x\geqq 0\};$
animal	$\rightarrow$ {pig};
nutrient	$\rightarrow$ {protein, calcium};
material	$\rightarrow$ {standard, additive};
MIN	$\rightarrow$ {<pig, protein>$\rightarrow$6, <pig, calcium>$\rightarrow$4};
UCOST	$\rightarrow$ {standard$\rightarrow$ 1.2, additive$\rightarrow$ 3};
Q	$\rightarrow$ {<pig, standard>$\rightarrow$2, <pig, additive>$\rightarrow$0.5};

ANALYSIS $\rightarrow$ {<protein, standard>$\rightarrow$ 4,
<protein, additive>$\rightarrow$ 14,
<calcium, standard>$\rightarrow$ 2,
<calcium, additive>$\rightarrow$ 1};

TOTCOST $\rightarrow$ $\big[$a$\in\mu$(animal)$\rightarrow$
SUM({μ(UCOST)(m)*μ(Q)(a,m) | m$\in\mu$(material)})$\big]$;

NLEVEL $\rightarrow$ $\big[$<a,n>$\in\mu$(animal)$\times\mu$(nutrient)$\rightarrow$
SUM({μ(ANALYSIS)(n,m)*μ(Q)(a,m) |
m$\in\mu$(material)})$\big]$;

T:NLEVEL $\rightarrow$ $\big[$<a,n>$\in\mu$(animal)$\times\mu$(nutrient)$\rightarrow$
$\begin{cases}\text{true if } \mu(\text{NLEVEL})(a,n) \geqq \mu(\text{MIN})(a,n); \\ \text{false otherwise}\end{cases}$ $\big]$,

(4) Extraction

Extraction is an operation to make a model for a subschema. Let M=<D,μ> be a model for a schema G, and G_S be a schema such that $G_S \subset G$. Then a model for G_S is called an extraction of M. We will show an extraction of Feedmix model, which is a model for the following schema G_S. G_S is the subschema which neglects nutritional requirements for feeds.

Subschema G_S :

pounds	= real | $\geqq$0;
dollars	= real | $\geqq$0;
material	= chr | true
UCOST	= [material : dollars];
Q	= [material : pounds];
TOTCOST	= [: dollars].

Extracted Model M_S=<D,μ_S>:

pounds	$\rightarrow$ {x | x$\in$R $\wedge$ x$\geqq$0};
dollars	$\rightarrow$ {x | x$\in$R $\wedge$ x$\geqq$0};
material	$\rightarrow$ {standard, additive};
UCOST	$\rightarrow$ {standard$\rightarrow$ 1.2, additive$\rightarrow$ 3};
Q	$\rightarrow$ {standard$\rightarrow$ 2, additive$\rightarrow$ 0.5};
TOTCOST	$\rightarrow$ *SUM*({μ(UCOST)(m)*μ(Q)(m) | m$\in\mu$(material)}).

5.2 Utilization of Operations

Operations for models and their combinations are utilized for various purposes. Join is used when we make a large scale model in a bottom up approach, i.e., we first make submodels and construct the whole model by joining them together. Projection and Extraction are used when a submodel for necessary parts is required. Enlargement is used for generalizing a model which is made under a particular situation, e.g., we can treat various kinds of animal feeds in the enlargement of Feedmix model.

6. Implementation

This section demonstrates some examples of modeling with MODEL/R(MOdel DEscription Language for Relational data) and its interpretation system, which are now under development on the constraint logic programming language CLP(R)[Heintze 87]. CLP(R) is a constraint programming language that has a capability of solving linear constraints, linear equations and inequalities, in the logic programming framework.

6.1 MODEL/R

We first show a description of Feedmix model by MODEL/R(Fig.3). A description of a model is called *Modelschema*, which is composed of four parts, *Type*, *Relation*, *Function* and *Data*. Type and Relation parts define a 'schema', Function and Data parts correspond to a 'model' for the 'schema'. Syntax of formulas in the function part are based on the relational calculus, and are similar to database query languages. In each formula, 'in' statement specifies the domain of variables used in the formula, and 'for' specifies the domain for the function 'sum'. The first formula means that 'totcost' is the summation of the value of ucost multiplied by the value of q for the same key value of materials. Extensional descriptions of relations are written in the data part. Details of the MODEL/R syntax and semantics are omitted here.

```
modelschema  'Feedmix'
      type
            nutrient     := chr#X;
            material     := chr#X;
            pounds       := real#X if X>=0;
            dollars      := real#X if X>=0
      relation
            min          :=  [nutrient]:[pounds];
            ucost        :=  [material]:[dollars];
            analysis     :=  [nutrient,material]:[pounds];
            q            :=  [material]:[pounds];
            nlevel       :=  [nutrient]:[pounds];
            t_nlevel     :=  [nutrient]:[boolean];
            totcost      :=  []:[dollars]
      function
            totcost#[]:[TC]:=
                  sum(U*Q, TC) in [ucost#[M]:[U], q#[M]:[Q]]
                        for [M] in material#M;

            nlevel#[N]:[NL]:=
                  sum(A*Q, NL) in [analysis#[N,M]:[A], q#[M]:[Q]]
                        for [M] in material#M;

            t_nlevel#[N]:[TN]:=
                  if(NL>=V, TN) in [min#[N]:[V], nlevel#[N]:[NL]]
      data
            material#[standard,additive];
            nutrient#[protein,calcium];
            min#[[protein]:[16],[calcium]:[4]];
            analysis#[
                  [calcium,standard]:[2],
                  [calcium,additive]:[1],
                  [protein,  standard]:[4],
                  [protein,  additive]:[14]];
            ucost#[[standard]:[1.2],[additive]:[3]];
            q#[[additive]:[0.5],[standard]:[2]]
end.
```

Fig.3 Feedmix Model

We will show an example of problem solving by the interpretation system of MODEL/R.

```
CLP(R) Version 2.02
(C) Copyright Monash University 1986

1 ?- [mbss].                    → consult the interpretation system

*** Yes ***

2 ?- load_model(feedmix).       → load Feedmix model

>> Feedmix

*** Yes ***

3 ?- solve(totcost#X,[],L).     → find the value of totcost

X = [] : [3.9]
L = [totcost # [] : [3.9]]

*** Retry *** ?

*** Maybe ***

4 ?- solve(t_nlevel#X,[],L).    → find the value of t_nlevel

X = [protein] : [false]
L = [nlevel # [protein] : [15], t_nlevel # [protein] : [false]]

*** Retry *** ? y               → backtrack to find other values

X = [calcium] : [true]
L = [nlevel # [calcium] : [4.5], t_nlevel # [calcium] : [true]]

*** Retry *** ?

*** Maybe ***
```

The predicate *solve* has three arguments. The first one is for specifying *the goal*. The second one is for specifying *input data*, which have the same meaning as definitions in Data part of Modelschema. Computed values during the execution are returned in the third argument. The predicate solve searches values for the goal in the following order.

1. in input data,
2. in Data part of Modelschema,
3. in computed values during the execution,
4. by evaluating the function.

When the function is evaluated, *subgoals* are generated if the function refers to other relations. It can be simply implemented by the backward inference mechanism of logic programming language. We illustrate the process of executing solve.

1. *Goal:* t_nlevel#X,
```
   Function  t_nlevel#[N]:[TN]:=
             if(NL>=V, TN) in [min#[N]:[V], nlevel#[N]:[NL]],
```

2. *Subgoal:* min#[N]:[V],
   min#[protein]:[16] is found in *Data part.*

3. *Subgoal:* nlevel#[protein]:[NL],
```
   Function nlevel#[protein]:[NL]:=
      sum(A*Q, NL) in [analysis#[protein,M]:[A], q#[M]:[Q]]
            for [M] in material#M;
   which is expanded in the form:
   Function nlevel#[protein]:[A1*Q1+A2*Q2]:=
      in [analysis#[protein,standard]:[A1], q#[standard]:[Q1]
           analysis#[protein,additive]:[A2],  q#[additive]:[Q2]];
```

4. *Subgoal:* analysis#[protein,standard]:[A1],
   analysis#[protein,standard]:[4] is found in *Data part.*

5. *Subgoal:* q#[standard]:[Q1],
   q#[standard]:[2] is found in *Data part.*

6. *Subgoal:* analysis#[protein,additive]:[A2],
   analysis#[protein,additive]:[14] is found in *Data part.*

7. *Subgoal:* q#[additive]:[Q2],
   q#[additive]:[0.5] is found in *Data part.*

8. Evaluate if(4*2+14*0.5>=16, TN),
   and obtain t_nlevel#[protein]:[false].

6.2 Examples of Modeling

(1) Critical Path Method

We show an example of problem solving for a cyclic model. *Cpm model* expresses the project network in Fig.4, which is in a sample program distributed with CLP(R). The number attached to each arc in Fig.4 denotes the time required for the task represented by the arc. The purpose of the critical path method is to find the earliest start time and the latest completion time for each node. 'if' statement in a formula specifies the condition for applying the formula. For example, if there exists a tuple [_,N]:[_] in relation 'path', then the first definition of 'es' will be applied, otherwise the second one will be applied.

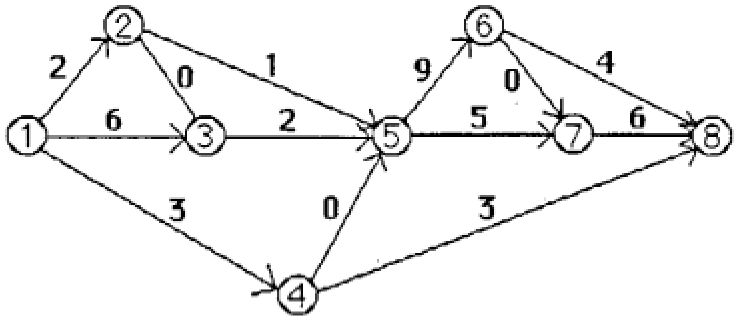

Fig.4 A Project Network Represented in Cpm Model

```
modelschema 'Cpm'
      type
            node := chr#X;
            time := real#X if X>=0
      relation
            es    := [node]:[time];          /* Earliest Start Time */
            lc    := [node]:[time];          /* Latest Completion time */
            path := [node,node]:[time]    /* Node Adjacency */
      function
            es#[N]:[Es] :=
                  max_(Es1+C,Es) in [path#[N1,N]:[C],es#[N1]:[Es1]]
                        for [N1] in path#[N1,N]:[_]
            if in path#[_,N]:[_];

            es#[N]:[0] if notin path#[_,N]:[_];

            lc#[N]:[Lc] :=
                  min_(Lc2-C,Lc) in [path#[N,N2]:[C],lc#[N2]:[Lc2]]
                        for [N2] in path#[N,N2]:[_]
            if in path#[N,_]:[_];

            lc#[N]:[T] := in es#[N]:[T] if notin path#[N,_]:[_]
      data
            node#[n1,n2,n3,n4,n5,n6,n7,n8];
            path#[
                  [n5,n6]:[9],[n5,n7]:[5],[n6,n7]:[0],[n4,n7]:[4],
                  [n1,n2]:[2],[n2,n3]:[0],[n1,n3]:[6],[n1,n4]:[3],
                  [n2,n5]:[1],[n2,n6]:[4],[n3,n5]:[2],[n4,n5]:[0],
                  [n4,n8]:[3],[n6,n8]:[4],[n7,n8]:[6]]

end.
```

Fig.5 Cpm Model

Earliest start time of node n7 and latest completion time of node n3 are computed as follows.

```
2 ?- load_model(cpm).

>> Cpm

*** Yes ***

3 ?- solve(es#[n7]:X,[],L).      → find the earliest start time for n7

X = [17]
L = [es # [n3] : [6], es # [n5] : [8], es # [n2] : [2], es # [n6] : [17],
es # [n1] : [0], es # [n4] : [3], es # [n7] : [17]]

*** Retry *** ?

*** Yes ***

4 ?- solve(lc#[n3]:Y,[],L).      → find the latest completion time of n3

Y = [6]
L = [lc # [n6] : [17], es # [n8] : [23], lc # [n8] : [23], lc # [n7] : [17],
lc # [n5] : [8], lc # [n3] : [6]]

*** Retry *** ?

*** Yes ***
```

(2) Simple Expert System

The next example is a simple expert system for determining next year's quota of sales in a book store. This example is described by an expert shell 'Guru' in [Holsapple 86]. In Adviser model, each if-then rule is transformed to a function for the data used in the rule. Therefore, consistency is required for the set of rules, i.e., there should exist no rules that give different values for one relation. In the following description, **read_data**() read the data from the user when it is evaluated.

```
modelschema  'Adviser'
     type
          money        := real#X;
          year         := int#X;
          economic_outlook:=  chr#X;
          prod         := chr#X;
          strength     := chr#X

     relation
          sales        := [year,prod]:[money];
          base         := [year,prod]:[money];
          quota        := [year,prod]:[money];
          economy      := [year]:[economic_outlook];
          growth       := [year]:[real];
          efactor      := [year]:[real];
          lafactor     := [year]:[real];
          localads     := [year]:[real];
          unemployment  :=  [year]:[real];
          pfactor      :=  [year,prod]:[real,strength];
          newtitles    := [year,prod]:[real];
          oldtitles    := [year,prod]:[real];
          rise         := [year]:[real];
          fall         := [year]:[real]

     function
          /* R1
           In case where the sales for this product exceeded the quota by more
           than 15%, the base amount for the new quota is set to the past quota
           plus the excess sales amount.
          */
          base#[Y,PR]:[Q+S-1.15*Q]
          if in [quota#[Y-1,PR]:[Q],sales#[Y-1,PR]:[S]]
               where S>1.15*Q;

          /* R2
          The base amount for the new quota is the same as the past quota
          because this product's sales did not exceed the past quota by more than
          15%.
          */
          base#[Y,PR]:[Q]
          if in [quota#[Y-1,PR]:[Q],sales#[Y-1,PR]:[S]]
               where S<=1.15*Q;
```

```
/* R3
When the local economic outlook is good, the economic factor is equal
to the economy's anticipated growth rate.
*/
efactor#[Y]:[G] := in growth#[Y]:[G]
if in economy#[Y]:[good];

/* R4-1
When the local economic outlook is fair, the economic factor is one
third of the growth rate.
*/
efactor#[Y]:[G/3] := in growth#[Y]:[G]
if in economy#[Y]:[fair];

/* R4-2
When the local economic outlook is fair, the local advertising factor is
1/120,000th of the amount budgeted for local advertising.
*/
lafactor#[Y]:[L/120000] := in localads#[Y]:[L]
if in economy#[Y]:[fair];

 /* R5
If the local economic outlook is poor, then economic factor should be
the lesser of the growth rate and the result of subtracting the
unemployment.
*/
efactor#[Y]:[E] := call((U1=0.85-U,G>U,E=U;E=G))
      in [growth#[Y]:[G],unemployment#[Y]:[U]]
if in economy#[Y]:[poor];

/* R6,7
The economic outlook is good because the projected unemployment
rate is below 7.6% and the anticipated growth rate is at least 4%, or
projected unemployment is less than 5.5% and the anticipated growth
rate is between 2%~4%.
*/
economy#[Y]:[good]
if in [growth#[Y]:[G],unemployment#[Y]:[U]]
      where (G>=0.04,U<0.076;G>=0.02,G<0.04,U<0.055);

/* R8
The economic outlook is fair because of moderate growth and
unemployment expectations.
*/
economy#[Y]:[fair] :=
if in [growth#[Y]:[G],unemployment#[Y]:[U]]
      where (G>=0.02,G<0.04,U>=0.055,U<0.082);
```

```
/* R9
The economic outlook is poor because either the anticipated growth
rate is very low or projected unemployment is high or both.
*/
economy#[Y]:[poor] :=
if in [growth#[Y]:[G],unemployment#[Y]:[U]]
      where (G<0.02;U>=0.082);

/* R10
When the economy is good and local advertising exceeds $2,000, the
local advertising factor is 1% for every thousand dollar expenditure.
*/
lafactor#[Y]:[L/100000]
if in [economy#[Y]:[good],localads#[Y]:[L]] where L>2000;

/* R11
When the economic outlook is poor and local advertising expenditures
for the product are modest, then the local advertising factor is
negative.
*/
lafactor#[Y]:[-0.015]
if in [economy#[Y]:[good],localads#[Y]:[L]] where L<1500;

/* R12
The local advertising factor is negligible because of low advertising
in a good economy or a poor economy coupled with substantial local
advertising for the product line.
*/
lafactor#[Y]:[0] :=
if in [economy#[Y]:[E],localads#[Y]:[L]]
      where (E=poor,L>=1500;E=good,L<=2000);

/* R13
This is a strong product line. The product factor is based on the
growth in the number of titles in this line.
*/
pfactor#[Y,PR]:[(NT+OT)/OT-1,strong] :=
      in [newtitles#[Y,PR]:[NT],oldtitles#[Y,PR]:[OT]]
if  member(PR,[computer,romance,scifi]);
```

```
/* R14
This is neither a strong nor weak product line. The product factor is
proportional to three fourths of the growth in the number of titles in
this line.
*/
pfactor#[Y,PR]:[0.75*((NT+OT)/OT-1),weak] :=
      in [newtitles#[Y,PR]:[NT],oldtitles#[Y,PR]:[OT]]
if  member(PR,[reference,biography,psychology,sports]);

/* R15
This is a strong product line. The base amount, economic factor, and
local advertising factor for calculating the new quota are all known.
A subjective assessment of the expected sales increase due to general
rising interest in the product is required. The new quota is then
calculated.
*/
quota#[Y,PR]:[B*(1+E+LA+P+R/100)] :=
      in [base#[Y,PR]:[B],efactor#[Y]:[E],lafactor#[Y]:[LA]]
      where read_data(rise#[Y]:[R])
if in pfactor#[Y,PR]:[P,strong];

/* R16
This is a weak product line. A subjective assessment of the expected
sales decrease due to general declining interest in this product line is
requested. The new quota can then be calculated.
*/
quota#[Y,PR]:[B*(1+E+LA+P-F/100)] :=
      in [base#[Y,PR]:[B],efactor#[Y]:[E],lafactor#[Y]:[LA]]
      where read_data(fall#[Y]:[F])
if in pfactor#[Y,PR]:[P,weak];

/* R17
This is neither an especially strong nor weak product line. Its new
quota is calculated from the base amount and factors for the economy,
local advertising, and product line expansion.
*/
quota#[Y,PR]:[B*(1+E+LA+P)] :=
      in [base#[Y,PR]:[B],efactor#[Y]:[E],lafactor#[Y]:[LA]]
if in pfactor#[Y,PR]:[P,medium];

/* R18
This is a weak product line. The product factor is proportional to less
than half of its growth in titles.
*/
pfactor#[Y,PR]:[0.45*((NT+OT)/OT-1),medium] :=
      in [newtitles#[Y,PR]:[NT],oldtitles#[Y,PR]:[OT]]
if  not(member(PR,[computer,romance,scifi,reference,
      biography,psychology,sports]))
```

```
        data
                year#[1987,1988];
                prod#[computer,romance,scifi,reference,
                        biography,psychology,sports];
                economic_outlook#[good,fair,poor];
                strength#[strong,weak,medium];
                sales#[[1987,computer]:[100],[1987,scifi]:[80]];
                quota#[[1987,computer]:[90],[1987,scifi]:[130]];
                growth#[[1988]:[0.03]];
                unemployment#[[1988]:[0.04]];
                localads#[[1988]:[3240]];
                newtitles#[[1988,computer]:[7],[1988,scifi]:[6]];
                oldtitles#[[1988,computer]:[17],[1988,scifi]:[16]]
end.
```

Quota of computer in 1988 is computed as follows.

```
2 ?- load_model(rules).

>> Adviser

*** Yes ***

3 ?- solve(quota#[1988,computer]:X,[],L).
                                → find the quota for computer in 1988
>> Read rise # [1988]:[20].     → Read rise(%) in 1988,
                                where rise is a subjective assessment of
                                the expected sales increase.

X = [150.675]
L = [lafactor # [1988] : [0.0324], economy # [1988] : [good],
efactor # [1988] : [0.03], base # [1988, computer] : [90],
pfactor # [1988, computer] : [0.411765, strong],
quota # [1988, computer] : [150.675]]

*** Retry *** ?

*** Yes ***
```

7. Conclusion

In this paper, we have formalized a model as a mapping from a relational data structure to a set of possible values in the real world. A model has been described as a set of functions, each of which represents the functional dependency in each relation. In addition, operations for models have been defined. These operations enables various kinds of approaches for modeling. Capabilities of this formalization have been demonstrated by MODEL/R and its interpretation system on a constraint logic programming language CLP(R). This system is one of the realizations for implementing MODEL/R. Other realizations are possible by using appropriate constraint solvers.

This research is a part of the work in the major R&D of the Fifth Generation Computer Project, conducted under the program set up by MITI.

Note. Research use of *CLP(R) interpreter version 2.0* in our organization(IIAS-SIS, FUJITSU LIMITED) is authorized by Monash University.

ACKNOWLEDGEMENTS

I thank to Dr.M.TODA, IIAS-SIS, for many valuable discussions and suggestions.

REFERENCES

[Blanning 86] R.W.Blanning: An Entity-Relationship Approach to Model Management, Decision Support Systems 2, 65/72 (1986).

[Bonczek 81] R.H. Bonczek, C.W. Holsapple and A.B. Whinston: A Generalized Decision Support System Using predicate Calculus and Network Data Base Management, Operations Research, 29-2, 263/281 (1981).

[Codd 74] E.F.Codd: Recent Investigations in Relational Data Base Systems", Information Processing 74, 1017/1021, North-Holland (1974).

[Dolk 1984] D. Dolk and B. Konsynski: Knowledge Representations for Model Management Systems, IEEE Trans. on Software Engineering, 10(6), 619/628 (1984).

[Elam 80] J.J. Elam, J.C. Henderson and L.W. Miller: Model Management Systems: An Approach to Decision Support in Complex Organizations, Proc. of the 1st Conference on Information Systems (1980).

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

[Heintze 87] N.C. Heintze et. al.: Constraint Logic Programming, 4th IEEE Symp. on Logic Programming (1987).

[Holsapple 86] C.W. Holsapple and A.B. Whinston: Manager's Guide To Expert Systems Using Guru, DowJones-Irwin, Illinois, U.S.A (1986).

[Leler 88] W. Leler: Constrained Programming Languages, Addison-Wesley (1988).