DATABASE SYSTEM BASED ON INTENSIONAL LOGIC

Naoki Yonezaki, Hajime Enomoto

DEPARTMENT OF COMPUTER SCIENCE
TOKYO INSTITUTE OF TECHNOLOGY
2-12-1, ŌOKAYAMA, MEGURO-KU
TOKYO 152, JAPAN

Model theoretic semantics of database systems is studied. As Rechard Montague has done in his work,[5] we translate statements of DDL and DML into intensional logic and the latter is interpreted with reference to a suitable model. Major advantages of its approach include (i) it leads itself to the design of database systems which can handle historical data, (ii) it provides with a formal description of database semantics.

1. INTRODUCTION

There have been developed several knowledge-base systems which utilize mathematical logic, however they can treat facts or rules at the current world only.[1~3]

In the medical fields or the area of artificial intelligence there are many applications in which the database systems with historical data handling capability are required. For example, in a query to a medical database 'Has a sterum treatment been applied to John?' historical data is essential.

One of the reasons why existing database systems provide poor support for such historical information is probably because very few theoretical study has been done yet.

Intensional logic (IL) which Rechard Montague developed to describe semantics of natural language formally seems to be useful to the theory of such database. The first application of modal logic to the logic of database was done by Lipski, though he treated incompleteness of database only.[4]

In the Montague's approach, concepts of intension and extension are used.[5,6] The same concepts can be applied in the theory of database. Correspondence between IL and semantics of database is shown in Fig.1.

[Intensional Logic] :	[Database]
possible worlds :	states of database
extension :	data at some state
intension :	historical data
meaning postulates :	integrity constraints

Fig.1 Correspondence between Intensionals Logic and Database system

The extension of a constant predicate P in IL which corresponds to a relation in database corresponds to the contents in the current database state.

The main purpose of this paper is to describe in a implementation-independent way aspects of those database semantics, which are characterized by interpretation of update operations and queries . And we show the feasibility of using intensional logic for description of the semantics.

The treatment of update of database is closely related to that of assignment in programming language with data type specification facility.

Firstly we use Montague's intensional logic and later we will introduce two-sorted type theory to treat queries which refer state indirectly.

In section 2, we define a data model treated in this article. This data model is considered as hierachical relational model. In section 3, syntax of intensional logic is defined and its semantics is stated in section 4. In section 5, 6, syntax of query statements and their Montague semantics are given. In section 7, 8, data manipulation statements are introduced and their semantics is also defined. In section 9, semantics of two kinds of null value is stated as meaning postulates. In section 10, we introduce two-sorted type theory and give semantics of statements referring states. Section 11 is a concluding section.

2. DATA MODEL

In this section we define a data model which corresponds to relational model exploiting hierarchical structure of relations, that is, each component of relation may be also a relation or set of relations recursively. In the Relational Model which Codd[7] introduced, 3rd-normal form or 4th-normal form is exploited to avoid the update anomalies, though we regard it as an implementational matter. Hierachical structure of relation is quite natural for representing information in the real world.

As a part of a data definition language (DDL), schema declaration is formally defined as follows. This DDL describes hierachical schema of database, name of each relation and attribute names or selector names of a relation. When we consider schema of relation, we do not concern the name of relation.

Let S_0 be the set of all elementary data types e.g. integer, real or string of characters and so on, and F be the set of selector names. Schemas of database are constructed from S_0 recursively as strings on $C_t = S_0 \cup \{[,],',:\} \cup F$.

Def. The set S of schemes is the smallest set S_2 satisfying (1),(2).

(1) $S_1 \supset S_0 \cup S_2$,

(2) $t_1,\ldots,t_n \in S_1$, $s_1,\ldots,s_n \in F$ ($s_i \neq s_j$ for $i \neq j$)
$\Rightarrow [s_1:t_1,\ldots,s_n:t_n] \in S_2$ $(1 \leq n)$.

Schema declaration in our DDL is of the form

Relation name = t

, where $t \in S$.

Now we can define the hierarchical relational database (HRDB) as follows.

Def. The set HD of HRDB is defined as

$$HD = \bigcup_{t \in S} \mathbf{D}_t \quad .$$

For $t \in S$, $\mathbf{D}_t$ is the set of database whose schema is t defined recursively with the following rules (1), (2).

(1) $t \in S_0 \Rightarrow \mathbf{D}_t = \mathbf{E}_t$, where $\mathbf{E}_t$ is the set of data having elementary type t,

(2) $t \in S$ and $t = [s_1:t_1,\ldots,s_n:t_n] \Rightarrow$
$\mathbf{D}_t = 2^{\mathbf{D}_{t_1}' \times \mathbf{D}_{t_2}' \times \ldots \times \mathbf{D}_{t_n}'}$
, where $\mathbf{D}_{t_i}' = \mathbf{D}_{t_i} \cup \{NULL1, NULL2\}$.

An example of a database is shown as follows.

Example 1

Now, we consider an employee relation comprising of tuples which have hierarchical structure. Each such tuple consists of employee number, name, education relation, age, a set of skills and children relation.

Corresponding schema declaration is as follows.

```
EMP1=[$E:int, NAME:string,
  EDUCATION:[SCHOOL:string, DEG:string, YR:int],
  SKILL:[SNAME:string],
  KIDS:[KNAME:string, AGE:int, SEX:string],
  SAL:int]
```

Fig.2 is an instance of database with this schema at some state or world.

EMP1

		EDUCATION			SKILL	KIDS			
$E	NAME	SCHOOL	DEG	YR	SNAME	KNAME	AGE	SEX	SAL
1	CARY	A B	NULL2 B	58 64	SA SE SM	JACK JILL JOHN	8 5 10	M F M	15K
2	JONES	C D A	A C NULL2	72 74 80	NULL1	NULL1			14K
3	SMITH	C	A	50	SB SD SE	MARY	17	F	20K
4	JACK								

Fig.2 Instance of the Employee Relation

The value 'NULL1' means nothing, however in some state there may be some values. The value 'NULL2' means absolutely no value exists in any state. The order of selector in each tuple is insignificant and order of raw is also insignificant. This hierarchical schema can be visualized by tree graph. Fig.2 is a two-dimensional representation of hierarchical data which reflects the images of data instances.

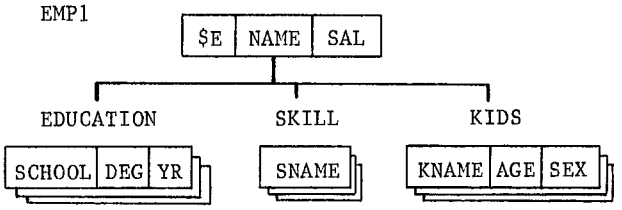

Fig.3 A tree graph of hierarchical data

Note that the data declaration in our DDL and this graph show the data structure only while the graph provides a convenient means to visualize the instances.

3. SYNTAX OF INTENSIONAL LOGIC

Intensional logic used in the Montague's approach is based on the theory of types. In this section we describe the extended version of IL according to Gallin.[8] The set of all possible types is recursively defined as follows.

Def. Let e, t, s be any three objects. The set of types of IL is the smallest set T satisfying (1)~(3).

(1) $e, t \in T$,
(2) $a, b \in T \Rightarrow \langle a, b \rangle \in T$,
(3) $a \in T \Rightarrow \langle s, a \rangle \in T$.

Objects of type e correspond to entities. Type of IL corresponds schema of our DDL or DML. Our schema supports n-ary relations, and it is considered as n-ary function whose range is truth values. For any function of two (or more) arguments there exists an equivalent one which takes one argument at a time,[9] i.e. $(D_1 \times D_2 \times \ldots \times D_n) \rightarrow D_{n+1}$ corresponds to $(D_1 \rightarrow (D_2 \rightarrow (\ldots \rightarrow (D_n \rightarrow D_{n+1})))) \ldots)$. In this way we can make the types in IL correspond to schemas of our DDL. Dtype which is a subset of type T in IL is now introduced as follows.

Def. Dtype is the smallest set of T_A satisfying (1)~(4).

(1) $e \in T_B$,
(2) $a \in T_A \Rightarrow \langle s, a \rangle \in T_B$,
(3) $a \in T_B \Rightarrow \langle a, t \rangle \in T_A$,
(4) $b \in T_B$, $a \in T_A \Rightarrow \langle b, a \rangle \in T_A$.

Constants of type $\langle s,a \rangle$ ($a \in$ Dtype) correspond to relations with schema corresponding to type a. Such translation will be defined formally in section 6. Intuitively speaking, constants of type $\langle s,\langle e,t \rangle \rangle$

corresponds to relations with schema [s: int], [s: real] or [s: string], i.e. single flat domain relation. Constants of type <s,<<s,a>,t>> correspond to relations with schema [s: [...]] i.e. hierarchical relations of single domain whose value is not flat but a relation with schema corresponding to type a. By rule (4), we can define the type of IL corresponding to n-ary relation schemas.

Example 2

The type of constant in IL which corresponds to relation in Example 1 is

$$\langle s,\langle e,\langle e,\langle\langle s,\langle e,\langle e,\langle e,t\rangle\rangle\rangle\rangle,\langle\langle s,\langle e,t\rangle\rangle \\ ,\langle\langle s,\langle e,\langle e,\langle e,t\rangle\rangle\rangle\rangle,\langle e,t\rangle\rangle\rangle\rangle\rangle\rangle\rangle.$$

We take CON_a (VAR_a) to be the set of constants (variables) of type a. Now, we define the set Tm_a of terms of IL of type a as follows.

Def.

(1) $CON_a \subset Tm_a$,

(2) $VAR_a \subset Tm_a$,

(3) $A, B \in Tm_e \Rightarrow A+B, A-B, A*B, A \div B \in Tm_e$,

(4) $A \in Tm_{<a,b>}, B \in Tm_a \Rightarrow A[B] \in Tm_b$,

(5) $A \in Tm_b, x \in VAR_a \Rightarrow \lambda x(A) \in Tm_{<a,b>}$,

(6) $A, B \in Tm_a \Rightarrow (A=B) \in Tm_t$,

(7) $A \in Tm_a \Rightarrow {}^{\wedge}A \in Tm_{<s,a>}$,

(8) $A \in Tm_{<s,a>} \Rightarrow {}^{\vee}A \in Tm_a$,

(9) $A, B \in Tm_a, P \in Tm_t \Rightarrow (P \rightarrow A, B) \in Tm_a$,

(10) $A \in Tm_a, c \in CON_{<s,b>}, B \in Tm_b \Rightarrow \{B/{}^{\vee}c\}A \in Tm_a$,

(11) $A \in Tm_t \Rightarrow PA \in Tm_t, FA \in Tm_t$.

The additional construct (10) is introduced by Janssen.[10] Following Henkin,[11] we define sentential connectives, quantifiers and modal operators as follows.

Def.

(1) $\mathbf{T} = [\lambda x_t x_t = \lambda x_t x_t]$,

(2) $\mathbf{F} = [\lambda x_t x_t = \lambda x_t \mathbf{T}]$,

(3) $\sim = \lambda x_t[\mathbf{F}=x_t]$,

(4) $\wedge = \lambda x_t \lambda y_t[\lambda f_{<t,t>}[fx=y]=\lambda f_{<t,t>}[f\mathbf{T}]]$,

(5) $\rightarrow = \lambda x_t \lambda y_t[[x \wedge y]=x]$,

(6) $\vee = \lambda x_t \lambda y_t[\sim x \rightarrow y]$,

(7) $\forall x_a A = [\lambda x_a A = \lambda x_a \mathbf{T}]$,

(8) $\exists x_a A = \sim \forall x_a \sim A$,

(9) $[A_a \underline{\Delta} B_a] = [{}^{\wedge}A_a = {}^{\wedge}B_a]$,

(10) $\Box A = [A \underline{\Delta} \mathbf{T}]$,

(11) $\Diamond A = \sim \Box \sim A$.

We write [A∧B] instead of [[∧A]B] where A and B are formulas, similarly for the other binary connectives.

4. SEMANTICS OF IL

The terms of intensional logic are interpreted in an intensional model. Such a model is a system $M = (M_a, <, m)_{a \in T}$, where M_a is a frame based on non-empty sets D and I, and < is a linear ordering on S. To simplify following arguments, we make no distinction among 'int', 'real' and 'string', and they are represented by D.

Def. Frame is defined as the indexed family $(M_a)_{a \in T}$ of sets, where

(1) $M_e = D \cup \{NULL1, NULL2\}$,

(2) $M_t = 2 = \{0,1\}$,

(3) $M_{<a,b>} = M_b^{M_a} = \{F \mid F: M_a \rightarrow M_b\}$,

(4) $M_{<s,a>} = M_a^{I} = \{F \mid F: I \rightarrow M_a\} \cup \{NULL1, NULL2\}$.

Function m must be such that if c_a is a constant of type a, then $m(c_a) \in M_{<s,a>}$.

The assignment g is a function from variable to value such that if $x \in VAR_a$ then $g(x_a) \in M_a$. If x_a is a variable of type a, and $x \in M_a$ then $g(x/X)$ denotes the value assignment exactly like g except that it assigns the value X to the variable x.

Now we define the interpretation of a term A_a in a model M with respect to the state s and the assignment g. It is denoted by $V_{M,s,g}(A_a)$ and defined inductively as follows. (We dropped the subscript 'M'.)

Def.

(1) $V_{s,g}(x_a) = g(x_a)$, $x_a \in VAR_a$,

(2) $V_{s,g}(c_a) = m(c_a)(s)$, $c_a \in CON_a$,

(3) $V_{s,g}(A_e + B_e) = V_{s,g}(A_e) + V_{s,g}(B_e)$ and similar for the other arithmetical operations,

(4) $V_{s,g}(A_{<a,b>}[B_a]) = V_{s,g}(A_{<a,b>})[V_{s,g}(B_a)]$,

(5) $V_{s,g}(\lambda x_a A_b)$ = the function f with domain M_a, such that whenever $X \in M_a$ then $f(x) = V_{s,g'}(A_b)$, where $g' = g(x/X)$. By using meta-operator Γ, we express the above as $\Gamma\, X[V_{s,g'}(A_b)]$,

(6) $V_{s,g}(A_a = B_a) = 1$ if $V_{s,g}(A_a) = V_{s,g}(B_a)$, and 0 otherwise,

(7) $V_{s,g}({}^{\wedge}A_a) = \Gamma\, j[V_{j,g}(A_a)]$,

(8) $V_{s,g}({}^{\vee}A_{<s,a>}) = V_{s,g}(A_{<s,a>})(s)$,

(9) $V_{s,g}((P_t \rightarrow A_a, B_a)) = V_{s,g}(A_a)$ if $V_{s,g}(P_t) = 1$, and $V_{s,g}(B_a)$ otherwise,

(10) $V_{s,g}(\{B_b/{}^{\vee}c_{<s,b>}\}A_a) = V_{t,g}(A_a)$, where $t = \langle c \leftarrow V_{s,g}(B_b) \rangle s$,

(11) $V_{s,g}(PA_t) = 1$ if there is some s' in S such that $s' < s$ and $V_{s',g}(A) = 1$, 0 otherwise,

(12) $V_{s,g}(FA_t) = 1$ if there is some s' in S such that $s < s'$ and $V_{s',g}(A) = 1$, 0 otherwise.

In the rule (10) t denotes the state in which all constants have values exactly like in s except that the value of ^{V}c is the value of the expression B_b in the state s.

We must clarify the notion of state in our model to define the meaning of database. State is understood to present the internal situation of a database. A state is altered by update operations. A new state s_1 preceeds an old state s_2, i.e. $(s_2<s_1)$. Clearly a state $s \in I$ determines the value of all relations. The effect of update operation is that it modifies the value of single relation, and in order to model this update operation, it should be assumed that the resulting state always exists and is unique.

So we restrict our model for IL which satisfies the following postulates.

1) Update postulate

For every $t_1 \in I$, every $c \in CON_{<s,a>}$ ($a \in$ Dtype) and every $i \in M_a$, there exists a unique $t_2 \in S$ such that

$$\begin{cases} V_{s,g}(c)(t_1) = i, \\ V_{s,g}(c')(t_1) = V_{s,g}(c')(t_2), \end{cases}$$

for all constants $c' \neq c$.

2) Uniqueness postulate

For every t_1, $t_2 \in I$ $(t_1 \neq t_2)$, there exists $c \in CON_{<s,a>}$ ($a \in$ Dtype) such that

$$V_{s,g}(c)(t_1) \neq V_{s,g}(c)(t_2),$$

that is, the result of update operation should be distinct from all of the other states.

Therefore, we can identify a state with contents of database. The database is translated into constant in IL. So the set I of states is defined by

$$I = \prod_{a \in Dtype} \prod_{c \in CON_{<s,a>}} M_a .$$

5. QUERY STATEMENTS

Now we introduce a syntax of queries in our data manipulation language (DML). The set Q of queries is defined as $Q_1 \cup Q_2$ recursively.

Def.

(1) $\mathbf{0}, \mathbf{1} \in Q_1$,

(2) d ∈ descriptor => $d \in Q_1$,

(3) $q_1, q_2 \in Q_1$ => $\neg q_1$, (q_1+q_2), $(q_1 \circ q_2)$, $(q_1 \rightarrow q_2)$, always q_1, past q_1, future $q_2 \in Q_1$,

(4) $f \in Q_2$, $q \in Q_1$ =>
q when f, q whenever $f \in Q_1$,

(5) $T, F \in Q_2$,

(6) $q_1, q_2 \in Q_1$ => $q_1=q_2 \in Q_2$,

(7) $f_1, f_2 \in Q_2$ => $\neg f_1$, $(f_1 \vee f_2)$, $(f_1 \wedge f_2)$, $(f_1 \Rightarrow f_2)$, always f, past f, future f,
f_2 when f_1, f_2 whenever $f_1 \in Q_2$.

Q_1 is the set of queries for which an answer is a set of objects. For a query in Q_2 an answer is yes or no.

Every descriptor is of the form <R, $\mathbf{s}_1$, $\mathbf{s}_2$, A>, where R is a relation name, $\mathbf{s}_1$, $\mathbf{s}_2$ are sequences of selectors, and A is the subset of $\mathbf{D}_t$ which is accessed by $\mathbf{s}_2$.

<R, $\mathbf{s}_1$, $\mathbf{s}_2$, A> is used to denote the set of $\mathbf{s}_1$ element of tuples in the relation R whose $\mathbf{s}_2$ element is in A. In the above rule (3) all descriptors in (q_1+q_2), $(q_1 \circ q_2)$ or $(q_1 \rightarrow q_2)$ must have the same selector sequence $\mathbf{s}_1$ as their second component.

We assume certain auxiliary language for describing A which is a subset of the domain $\mathbf{D}_t$.

Example 3

Let us consider a query, 'Was every employee who has a child named Jack educated at school B or C?' on the database given in the previous example. We have a expression

<EMP, NAME, KIDS.KNAME, {Jack}> ∘
-<EMP, NAME, EDUCATION.SCHOOL, {B, C}>=**0**.

Let us consider another example as follows.
'Get all names of employees who always earn more than 10000 or less than 500.' The corresponding expression is

always(<EMP, NAME, SAL, {x|x>10000}> +
<EMP, NAME, SAL, {x|x<500}>).

6. SEMANTICS OF QUERIES

We have already defined a model theoretic meaning of IL. Thus the remaining part of Montague semantics is defining a translation which gives for each syntactic structure of the language some meaningful expression of IL. For a part of statement E, its translation into IL is denoted E'. Relation name R is translated into a constant R' of IL whose type is determined by its schema declared at DDL. More precisely this type is determined with next rules. If there is a statement 'Rname: t' in a data definition part, then type of the constant Rname' is G(t), where G is a function satisfying,

Def. G: S → Dtype

(1) $t = [s_1:t_1, s_2:t_2, \ldots, s_n:t_n]$ =>
$G(t)=F(t)=<s,<F(t_1),<F(t_2),\ldots,<F(t_n),t>>\ldots>$,

(2) $t \in S_0$ => $F(t) = e$.

Each descriptor <R, $\mathbf{s}_1$, $\mathbf{s}_2$, A> is translated according to the schema of R and sequences of selectors $\mathbf{s}_1$ and $\mathbf{s}_2$. Let $\mathbf{s}_0$ denote the common maximal prefix of $\mathbf{s}_1$, $\mathbf{s}_2$ and $|\mathbf{s}_0|^{\dagger}=n$, $|\mathbf{s}_1|=n+k$, $|\mathbf{s}_2|=n+q$, i.e. $\mathbf{s}_1$, $\mathbf{s}_2$ are represented as $\mathbf{s}_1=s_{01} \cdot s_{02} \cdots s_{0n} \cdot s_{11} \cdot s_{12} \cdots s_{1k}$, $\mathbf{s}_2=s_{01} \cdot s_{02} \cdots s_{0n} \cdot s_{21} \cdot s_{22} \cdots s_{2q}$.

† For string **s**, $|\mathbf{s}|$ denotes length of **s**.

$$[<R, s_1, s_2, A>]' =$$
$$\lambda i_{s_{1k}} (\exists i_{s_{0n}} ([R(s_{01}\ldots s_{0n})]'[R'][i_{s_{0n}}]$$
$$\wedge \exists i_{s_{11}} \exists i_{s_{21}} (\exists i_1 \ldots \exists i_m$$
$$(\vee i_{s_{0n}} [i_1]\ldots[i_{s_{11}}]\ldots[i_{s_{21}}]\ldots[i_m])$$
$$\wedge [R(s_{12}\ldots s_{1k})]'(i_{s_{11}})(i_{s_{1k}})$$
$$\wedge \exists i_{s_{2q}} ([R(s_{22}\ldots s_{2q})]'(i_{s_{21}})(i_{s_{2q}})$$
$$\wedge \vee A'[i_{s_{2q}}])))$$

, where the sequence $\exists i_1 \ldots \exists i_m$ does not contain $\exists i_{s_{11}}$ or $\exists i_{s_{21}}$. The number m and positions of $[i_{s_{11}}]$, $[i_{s_{21}}]$ in the sequence of $[i_1]\ldots[i_m]$ are determined self evidently by the DDL statement about R. If n=0 then the first component $[R(s_{01}\ldots s_{0n})]'[R'][i]$ of the above conjunctive form is replaced by R'=i. If k=0 or q=0 then an appropriate but obvious modification is also needed. It is possible to construct this IL expression systematically according to a micro syntax of a descriptor, however it is somewhat tedious so we do not concern it. When a is a type of sub-schema corresponding to the selector s_{21}, A' is a predicate of type <a, t> in which A'(i) has the value 1 when an object corresponding to i is an element of A.

Each i_s is a bounded variable whose type is the one determined by G and sub-schema corresponding to a selector s used in a relation R.

Translation of $R(s_1\ldots s_n)$ is defined as follows.

$$[R(s_1\ldots s_n)]' = \lambda i \lambda i_s (\exists i_1 \ldots \exists i_m (\vee i[i_1]\ldots[i_m] \wedge [R(s_2\ldots s_n)]'(i)(i_s)))$$
$$[R(s_n)]' = \lambda i \lambda i_s (\exists i_1 \ldots \exists i_m (\vee i[i_1]\ldots[i_s]\ldots[i_m]))$$

, where, each bounded variable $i_1,\ldots,i_m$ has the same type determined by G and sub-schema corresponding each selector which appears in the same level of relation R as the selector s_n appears. Moreover the sequence $\exists i_1,\ldots, \exists i_n$ does not contain $\exists i_s$.

According to the syntax of Q, translation into IL expression is defined as follows.

Def.

(1) $[\mathbf{0}]' = \wedge\lambda x[\mathbf{F}][x]$,

(2) $[\mathbf{1}]' = \wedge\lambda x[\mathbf{T}][x]$,

(3) $[-q]' = \wedge\lambda x(\sim(\vee[q]'[x]))$,

(4) $[q_1+q_2]' = \wedge\lambda x(\vee[q_1]'[x] \vee \vee[q_2]'[x])$,

(5) $[q_1 \circ q_2]' = \wedge\lambda x(\vee[q_1]'[x] \wedge \vee[q_2]'[x])$,

(6) $[q_1 \rightarrow q_2]' = \wedge\lambda x(\sim\vee[q_1]'[x] \vee \vee[q_2]'[x])$,

(7) $[\text{always } q]' = \wedge\lambda x(\Box\vee[q]'[x])$,

(8) $[\text{past } q]' = \wedge\lambda x(P\vee[q]'[x])$,

(9) $[\text{future } q]' = \wedge\lambda x(F\vee[q]'[x])$,

(10) [q when f]', [q whenever f]':
these will be defined in section 10,

(11) $[T]' = \wedge\mathbf{T}$, $[F]' = \wedge\mathbf{F}$,

(12) $[q_1 = q_2]' = \wedge(\vee[q_1]' = \vee[q_2]')$,

(13) $[\neg f]' = \wedge(\sim\vee[f]')$,

(14) $[f_1 \vee f_2]' = \wedge(\vee[f_1]' \vee \vee[f_2]')$,

(15) $[f_1 \wedge f_2]' = \wedge(\vee[f_1]' \wedge \vee[f_2]')$,

(16) $[f_1 \Rightarrow f_2]' = \wedge(\sim\vee[f_1]' \vee \vee[f_2]')$,

(17) $[\text{always } f]' = \Box\vee[f]'$,

(18) $[\text{past } f]' = P\vee[f]'$,

(19) $[\text{future } f]' = F\vee[f]'$,

(20) [f_2 when f_1]', [f_2 whenever f_1]':
these will also be defined in section 10.

In case of [**0**]', [**1**]', type of x is determined by the context of its usage in a query.

7. DATA MANIPULATION STATEMENTS

Insertion, deletion and update of tuples in database are now considered. We concern the case where we can manipulate either one tuple at a time or a set of tuples with a single command.

Syntax of update statements is defined as follows. Update statements are built up from certain descriptor and operation. More exactly, the set M of data manipulation statements is defined with the following four cases:

(1) (set oriented update)
d: descriptor, <f,d> ∈ M
, where f means arbitrary operation on the object in the answer of d,

(2) (individual insertion)
(2-1) <R, t> ∈ M
, where, R is a relation name and t is a tuple which is intended to be inserted into R,
(2-2) d: descriptor, <d, t> ∈ M,

(3) (individual deletion)
(3-1) ~<R, t> ∈ M,
(3-2) d:descriptor, ~<d, t> ∈ M,

(4) (set oriented deletion)
~<R, S, A> ∈ M
, where, R is a relation name, $\mathbf{s}_2$ is a sequence of selectors and A is the subset of $\mathbf{D}_t$ which is accessed by $\mathbf{s}_2$.

A set oriented update statement (1) changes all elements in a relation R that are elements of answer for query q by values which is a result of operation f on them. An individual insertion (2-1) means insertion of a tuple t into a relation R as usual, and an individual insertion (2-2) means insertion of a tuple t into the all relations which are elements of answer for query d. We can define the meaning of individual deletions (3-1), (3-2) in the same manner as in the case of insertion by changing the word 'insert' by 'delete'. Set oriented deletion statement (4) deletes all tuples whose values accessed by $\mathbf{s}_2$ are in A from relation R.

8. SEMANTICS OF DATA MANIPULATION

In the Montague semantics, a data manipulation statement is translated into forward predicate transformer, which is a function from a state predicate to a state predicate whose type is <s, t>, and which has the format of an intension of an assertion. So the predicate transformer has a type <<s, t>, <s, t>> and has the format $\lambda P(\phi)$, where $P \in VAR_{<s,t>}$ and ϕ is a term of type <s, t>. For set oriented update statement, we define the translation into IL as follows. In this definition symbol '+' is used to designate the inverse of the translation, i.e. $[[E]^+]'=E$ for a term E of IL.

$$[<f,d>]'=[R:=[\lambda i_1 \ldots \lambda i_{s_{01}} \ldots \lambda i_m (\exists j_{s_{01}} (^\vee R'[i_1] \ldots [j_{s_{01}}] \ldots [i_m] \wedge [\%R(s_{02} \ldots s_{0n})]' [j_{s_{01}}][i_{s_{01}}]))]^+]',$$

$$[\%R(s_{02} \ldots s_{0n})]'=\lambda j \lambda i (i = {}^\wedge(\lambda i_1 \ldots \lambda i_{s_{02}} \ldots \lambda i_m (\exists j_{s_{02}} (^\vee j[i_1] \ldots [j_{s_{02}}] \ldots [i_m] \wedge [\%R(s_{03} \ldots s_{0n})]' [j_{s_{02}}][i_{s_{02}}]))))),$$

$$[\%R(s_{0n})]'=\lambda j \lambda i (i = {}^\wedge(\lambda i_1 \ldots \lambda i_{s_0} \ldots \lambda i_m (\exists j_{s_{0n}} (^\vee j[i_1] \ldots [j_{s_{0n}}] \ldots [i_m] \wedge i_{s_{0n}} = {}^\wedge(\lambda i_1 \ldots \lambda i_{s_{11}} \ldots \lambda i_{s_{21}} \ldots \lambda i_m (\exists j_{s_{11}} (^\vee j_{s_{0n}}[i_1] \ldots [j_{s_{11}}] \ldots [i_{s_{21}}] \ldots [i_m] \wedge ([\$R(s_{22} \ldots s_{2q})]' [i_{s_{21}}] \rightarrow [*R(s_{12} \ldots s_{1k})]'[j_{s_{11}}][i_{s_{11}}], i_{s_{11}} = j_{s_{11}})))))))),$$

$$[\$R(s_{22} \ldots s_{2q})]'=\lambda i (\exists i_1 \ldots \exists i_{s_{22}} \ldots \exists i_m (^\vee i[i_1] \ldots [i_{s_{22}}] \ldots [i_m] \wedge [\$R(s_{23} \ldots s_{2q})]'[i_{s_{22}}])),$$

$$[\$R(s_{2q})]'= \lambda i (\exists i_1 \ldots \exists i_{2q} \ldots \exists i_m (^\vee i[i_1] \ldots [i_{s_{2q}}] \ldots [i_m] \wedge {}^\vee A'[i_{s_{2q}}])),$$

$$[*R(s_{12} \ldots s_{1k})]'=\lambda j \lambda i (i = {}^\wedge(\lambda i_1 \ldots \lambda i_{s_{12}} \ldots \lambda i_m (\exists j_{s_{12}} (^\vee j[i_1] \ldots [j_{s_{12}}] \ldots [i_m] \wedge [*R(s_{13} \ldots s_{1k})]' [j_{s_{12}}][i_{s_{12}}])))),$$

$$[*R(s_{1k})]'=\lambda j \lambda i (i = {}^\wedge(\lambda i_1 \ldots \lambda i_{s_{1k}} \ldots \lambda i_m (\exists j_{s_{1k}} (^\vee j [i_1] \ldots [j_{s_{1k}}] \ldots [i_m] \wedge i_{s_{1k}} = f'(j_{s_{1k}}))))).$$

For individual insertion or deletion their translations are as follows.

$$[<R,t>]'=[R:=[\lambda i_1 \ldots \lambda i_m (i_1=t_1 \wedge \ldots \wedge i_m=t_m \rightarrow \mathbf{T}, R'[i_1] \ldots [i_m])]^+]',$$

$$[\sim<R,t>]'=[R:=[\lambda i_1 \ldots \lambda i_m (i_1=t_1 \wedge \ldots \wedge i_m=t_m \rightarrow \mathbf{F}, R'[i_1] \ldots [i_m])]^+]'$$

, where we assumed $t=<t_1, t_2, \ldots, t_m>$. If n=0, k=0 or q=0 then appropriate modifications are needed as in the case of query statements. For insertion operation <d,t> and deletion operations ~<d,t>; ~<R,S,A>, we can define their corresponding IL expressions in the same manner as shown in the case of set oriented update.

By using an intensional version of Floyd's semantics of assignment[12]

$$[A:=B]'=\lambda P {}^\wedge \exists z [\{z/{}^\vee A'\}^\vee P \wedge {}^\vee A' = \{z/{}^\vee A'\}B'],$$

and semantics of composition

$$[A;B]'= \lambda P [B'(A'(P))],$$

we complete the definition of the semantics of the update statements.

Now, we consider the following simple example for comprehension of the above definition.

Example 4

Suppose relation EMP2 is declared at data definition part as

EMP2=[KIDS:[NAME:string, AGE:int], SAL:int],

and that before the update,

<*2,<EMP2, SAL, KIDS.NAME, {Jack}>>

holds that $^\wedge([<EMP2, SAL, KIDS.NAME, \{Jack\}>]' = {}^\wedge \lambda x(x=20'))$.

We denote this precondition by PC. The translation of the update statement is

$$[<*2, <EMP2, SAL, KIDS.NAME, \{Jack\}>>]'$$
$$= [EMP2:=[\lambda i_K \lambda i_S (\exists j_S ({}^\vee EMP2'(i_K)(j_S) \wedge (\exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_A = Jack') \rightarrow i_S = j_S*2, i_S = j_S)))]^+]'$$
$$= \lambda P {}^\wedge \exists z (\{z/{}^\vee EMP2'\}^\vee P \wedge {}^\vee EMP2' = \{z/{}^\vee EMP2'\} (\lambda i_K \lambda i_S (\exists j_S ({}^\vee EMP2'(i_K)(j_S) \wedge (\exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_A = Jack') \rightarrow i_S = j_S*2, i_S = j_S))))).$$
$$PC = {}^\wedge({}^\wedge \lambda i_S (\exists i_K ({}^\vee EMP2'(i_K)(i_S) \wedge \exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_N = Jack'))) = {}^\wedge \lambda x(x=20')).$$

Then after the update,

$$^\wedge \exists z ({}^\wedge \lambda i_S (\exists i_K (z(i_K)(i_S) \wedge \exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_N = Jack'))) = {}^\wedge \lambda x(x=20')) \wedge {}^\vee EMP2' = (\lambda i_K \lambda i_S (\exists j_S (z(i_K)(j_S) \wedge (\exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_A = Jack') \rightarrow i_S = j_S*2, i_S = j_S))))) .$$

From this we derive that,

$$^\wedge \lambda i_S (\exists i_K ({}^\vee EMP2'(i_K)(i_S) \wedge \exists i_N \exists i_A ({}^\vee i_K(i_N)(i_A) \wedge i_N = Jack'))) = {}^\wedge \lambda x(x=40').$$

Example 5

As another example, we consider the relation as follows.

EMP3=[NAME: string, MGR: string].

For this relation we perform an update such as the manager of manager of John is Smith, i.e.

<=Smith,<EMP3,MGR,NAME,<EMP3,MGR,NAME,{John}>>>.

The translation of this update statement is

[<=Smith, <EMP3, MGR, NAME, <EMP3, MGR, NAME, {John}>>>]'

$$=[EMP3:=[\lambda i_N \lambda i_M(\exists j_M(^{\vee}EMP3'(i_N)(j_M)$$
$$\wedge(^{\vee}[<EMP3, MGR, NAME, \{John\}>]'(i_N)$$
$$\rightarrow j_M{=}Smith', j_M{=}i_M)))]^{+}]'$$
$$=[EMP3:=[\lambda i_N \lambda i_M(\exists j_M(^{\vee}EMP3'(i_N)(j_M)$$
$$\wedge(^{\vee\wedge}\lambda i_{M2}(\exists i_{N2}(^{\vee}EMP3'(i_{N2})(i_{M2})$$
$$\wedge i_{N2}{=}John'))(i_N)$$
$$\rightarrow j_M{=}Smith', j_M{=}i_M)))]^{+}]'$$
$$=\lambda P^{\wedge}\exists z(\{z/^{\vee}EMP3'\}^{\vee}P\wedge^{\vee}EMP3'{=}$$
$$(\lambda i_N \lambda i_M(\exists j_M(z(i_N)(j_M)$$
$$\wedge(\exists i_{N2}(z(i_{N2})(i_N)\wedge i_{N2}{=}John')$$
$$\rightarrow j_M{=}Smith', j_M{=}i_M))))).$$

Assume that before the update holds that
$$^{\wedge}(([<EMP3, MGR, NAME, \{John\}>]'{=}^{\wedge}\lambda x(x{=}John'))$$
$$\wedge([<EMP3, MGR, NAME, \{Smith\}>]'{=}$$
$$^{\wedge}\lambda x(x{=}Jack'))).$$

Then we obtain that afterwards
$$^{\wedge}\exists z(z(John')(John')\wedge z(Smith')(Jack')$$
$$\wedge^{\vee}EMP3'{=}(\lambda i_N \lambda i_M(\exists j_M(z(i_N)(j_M)$$
$$\wedge(\exists i_{N2}(z(i_{N2})(i_N)\wedge i_{N2}{=}John')$$
$$\rightarrow j_M{=}Smith', j_M{=}i_M))))).$$

This implies
$$^{\vee}EMP3'(John')(Smith')\wedge^{\vee}EMP3'(Smith')(Jack').$$

9. SEMANTICS OF NULL VALUES

We can define various kinds of null values, indeed ANSI/SPARC interim report[13] cites 14 possible manifestations of null. However, the two important kinds of null value have the meanings 'value at present unknown' and 'property inapplicable'.

Formal treatment of the first kind of null value has been resolved by the ideas of null substitution principle and non-truth functionality principle.[4,14]

In this article we study the second kind of null value more precisely. We introduced two null values NULL1, NULL2 of the second kind of null values. NULL1 is intended to mean that nothing exists at current state but there may be some states in which the value exists. NULL2 is intended to mean that there exists absolutely no value in any state.

Those semantics are formally defined by auxiliary update postulates.

(1) Update postulate for NULL1

For every $t_1 \in I$ and every $A \in Tm_{<s,a>}$ $(a=<a_1,<\ldots<a_n,t>>\ldots> \in Dtype)$,

if $V_{s,g}(A)(t_1)(f_{a_1})\ldots(f_{a_k})\ldots(f_{a_n})=1$
and f_{a_k}=NULL1

,then $V_{s,g}(A)(t_1)(f_{a_1})\ldots(h_{a_k})\ldots(f_{a_n})=0$
for all $h_{a_k} \in M_{a_k}-\{NULL1\}$

and there exists $t_2 \in I$ such that

$V_{s,g}(A)(t_2)(f_{a_1})\ldots(r_{a_k})\ldots(f_{a_n})=1$
for $r_{a_k} \in M_{a_k}-\{NULL1, NULL2\}$.

(2) Update postulate for NULL2

For every $t_1 \in I$ and every $A \in Tm_{<s,a>}$ $(a=<a_1,<\ldots<a_n,t>>\ldots> \in Dtype)$,

if $V_{s,g}(A)(t_1)(f_{a_1})\ldots(f_{a_k})\ldots(f_{a_n})=1$
and f_{a_k}=NULL2

, then for every $t_2 \in I$

and every $h_{a_k} \in M_{a_k}-\{NULL2\}$

$V_{s,g}(A)(t_2)(f_{a_1})\ldots(h_{a_k})\ldots(f_{a_n})=0$.

10. SEMANTICS OF STATE REFERENCE

To define the semantics of queries with 'when' or 'whenever', it is necessary to consider the case that s is a type. We define such a logic called Two-Sorted Type Theory, and denote it by Ty_2.

Def. The set T' of types of Ty_2 is the smallest set such that
(1) e, t, s $\in$ T',
(2) a, b $\in$ T' => <a,b> $\in$ T'.
Note that T $\subset$ T' (T: the set of types of IL).

Def. The sets Tm'_a of terms of Ty_2 of type a are defined with the following rules recursively.

(1) $CON_a \subset Tm'_a$,
(2) $VAR_a \subset Tm'_a$,
(3) A, B $\in Tm'_e$ => A+B, A-B, A*B, A÷B $\in Tm'_e$,
(4) $A \in Tm'_{<a,b>}$, $B \in Tm'_a$ => $A[B] \in Tm'_b$,
(5) $A \in Tm'_b$ => $\lambda x_a A \in Tm'_{<a,b>}$,
(6) $A, B \in Tm'_a$ => $(A{=}B) \in Tm'_t$,
(7) $A, B \in Tm'_a$, $P \in Tm'_t$ => $(P \rightarrow A,B) \in Tm'_a$,
(8) $A \in Tm'_a$, $c \in CON_{<s,b>}$, $x \in VAR_s$, $B \in Tm'_b$
=> $\{B/c[x]\}A \in Tm'_a$,
(9) $A \in Tm'_t$ => $PA \in Tm'_t$, $FA \in Tm'_t$.

To define the semantics of Ty_2 the model $M'=(M_a, <, m)_{a \in T'}$ is introduced, where M_a is a frame based on non-empty sets D, I, and < is a linear ordering on I.

We define the frame as the indexed family $(M_a)_{a \in T'}$ of sets, where

Def.
(1) $M_e = D \cup \{NULL1, NULL2\}$,
(2) $M_t = 2 = \{0, 1\}$,
(3) $M_s = I$,
(4) $M_{<a,b>} = M_b{}^{M_a} = \{F \mid F: M_a \rightarrow M_b\}$ (for $a \neq s$),
(5) $M_{<s,a>} = M_a{}^I = \{F \mid F: I \rightarrow M_a\} \cup \{NULL1, NULL2\}$.

For each constant c_a, $m(c_a) \in M_a$, and for each variable x_a, $g(x_a) \in M_a$. Interpretation of term A_a which is denoted $V_{M,g}(A_a)$ is defined in such a way that the following conditions hold. (We dropped the subscript 'M'.)

Def.

(1) $V_g(x_a) = g(x_a)$, $x_a \in VAR_a$,

(2) $V_g(c_a) = m(c_a)$, $c_a \in CON_a$,

(3) ~ (7) similar to the case of IL,

(8) $V_g(\{B_b/c_{<s,b>}[x_s]\}A_a) = V_{g'}(A_a)$,
, where $g' = g(x_s/X)$, $x_s \in VAR_s$, $X \in I$,
such that for all constant $c^i_{<s,b>}$ except c,
$V_g(c^i[x_s]) = V_{g'}(c^i[x_s])$
and $V_{g'}(c[x_s]) = V_g(B)$,

(9) $V_g(PA_t) = 1$ if there is some $g' = g(x_s/X)$ such that $g'(x_s) < g(x_s)$ and $V_{g'}(A) = 1$,
0 otherwise,

(10) $V_g(FA_t) = 1$ if there is some $g' = g(x_s/X)$ such that $g(x_s) < g'(x_s)$ and $V_{g'}(A) = 1$,
0 otherwise.

For each term A_a of IL we can define the translation of A_a in Ty_2. Such translation is defined with next rules, and denoted A^*_a.

Def.

(1) $[x_a]^* = x_a$,

(2) $[c_a]^* = c_{<s,a>}[x_s]$,

(3) $[A_{<a,b>}[B_b]]^* = A^*[B^*]$,

(4) $[\lambda x_a A_b]^* = \lambda x A^*$,

(5) $[A_a = B_a]^* = [A^* = B^*]$,

(6) $[\hat{}A_a]^* = \lambda x_s A^*$,

(7) $[\check{}A_{<s,a>}]^* = A^*[x_s]$.

Now we can define the semantics of query 'q when f' as a term of Ty_2.

$$[q \text{ when } f]' = \lambda x^1_s \lambda x (\exists x^2_s ([[f]']^*[x^2_s] \wedge [[q]']^*[x^2_s][x]))$$

, which means intuitively the set of objects which satisfy q in each states where f has a value 1. Similarly we define the following.

$$[q \text{ whenever } f]' = \lambda x^1_s \lambda x (\forall x^2_s ([[f]']^*[x^2_s] \wedge [[q]']^*[x^2_s][x])),$$

$$[f_2 \text{ when } f_1]' = \lambda x^1_s \lambda x (\exists x^2_s ([[f_1]']^*[x^2_s] \wedge [[f_2]']^*[x^2_s])),$$

$$[f_2 \text{ whenever } f_1]' = \lambda x^1_s \lambda x (\forall x^2_s ([[f_1]']^*[x^2_s] \wedge [[f_2]']^*[x^2_s])).$$

11. CONCLUDING REMARKS

We believe we have demonstrated feasibility of the Montague's approach to the semantics of hierarchical database systems with historical data.

As for future reasearch directions, we would like to point out the importance of (i) finding normal forms which allows efficient evaluation of terms, and (ii) finding a set of axioms for equivalence transformations which derive normal forms from given terms.

ACKNOWLEDGEMENTS

Our thanks are due to Mr. Kenichi Murata for fruitful discussions and encouragement and to Prof. Takuya Katayama and many other people whose ideas we have unwittingly absorbed over the years.

REFERENCES

1. Gallaire, H., Minker, J., (1978) 'Logic and Databases', Plenum Press.
2. Reiter, R., (1977) 'An Approach to Deductive Question-Answering', BBN Report No.3649, Bolt, Beranek and Newman, Inc.
3. Wong, H.K.T., and Mylopoulos, J., (1977) 'Two Views of Data Semantics: A survey of Data Models in Artificial Intelligence and Database Management', INFOR, 15, 3, 344-383.
4. Lipski, W., Jr., (1977) 'On Semantic Issues Connected with Incomplete Databases', 3-rd VLDB.
5. Montague, R., (1973) 'The Proper Treatment of Quantification in Ordinary English' Approaches to Natural Language, Reidel Dordrecht.
6. Montague, R., (1977) 'Universal grammar', Formal philosophy-selected paper of R. Montague, edited by R.H. Tomason, Yale Univ. Press.
7. Codd, E.F., (1974) 'Recent investigations in relational database systems', Information Processing 74, North-Holland Pub. Co., Amsterdam, 1017-1021.
8. Gallin, D., (1975) 'Intensional and Higher-order Modal logic', North-Holland Publishing Company, Amsterdam.
9. Curry, H.B., and Feys, R.,(1968) 'Combinatory Logic', Vol.1, North-Holland, Amsterdam.
10. Yanssen, T.M.V., (1977) 'The expressive power of intensional logic in the semantics of programming language', Lecture Notes in Comp. Sci. 53. Springer-Verlag, Berlin.
11. Henkin, L., (1963) 'A theory of propositonal types', Fund. Math., 52.
12. Floyd, R. W., (1967) 'Assingning meanings to programs', Proc. Amer. Math. Soc. Symposia in Applied Mathematics, Vol. 19.
13. ANSI/X3/SPARC Study Group on Data Base Management Systems, (1975) 'Interim Report', ANSI.
14. Yannis Vassiliou, (1979) 'Null Values in Data Base Management: A Denotational Semantics Approach', Internal Conference on Management of Data, ACM-SIGMOD, 162-169.