LBL-12378
c.2

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Physics, Computer Science & Mathematics Division

Presented at the Seventh International Conference on Very Large Data Bases, Cannes, France, September 9-11, 1981; and to be published in the Proceedings

DATA MODELS AND DATA MANIPULATION LANGUAGES: COMPLEMENTARY SEMANTICS AND PROOF THEORY

Harry K.T. Wong

March 1981

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

LBL-12378
c.2

LBL-12378

Data Models and Data Manipulation Languages: Complementary Semantics and Proof Theory

Harry K. T. Wong

Computer Science and Mathematics Department
Lawrence Berkeley Laboratory
University of California, Berkeley, CA 94720

This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

Proceedings of the Seventh International Conference on Very Large Data Bases, September 1981.

Data Models and Data Manipulation Languages: Complementary Semantics and Proof Theory

Abstract

We present briefly a language which integrates the description of a data model, data manipulation language and integrity constraints into one coherent framework, resembling that proposed by several recent papers in the field of semantic data models. We then give two formal specifications of the semantics of the model and DML: one, based on states and state transactions, intended for database implementors and programmers, and one, based on axioms and partial correctness assertions intended for verifiers who wish to show that the system maintains integrity constraints. Most significantly, we sketch the proof that the deductive theory is sound and complete and hence "matches" the state transition semantics.

1. Introduction

The proliferation of data models in the last decade and especially the increased sophistication of the so-called semantic data models (see for example [Tsichritzis & Lochovsky81]) has led to problems which are reminiscent to those arising after the explosion in new programming languages during the 1960's. Because the description of the data models (DM's) is usually presented in an informal, English discussion, potential database designers may not be clear exactly what a correct database must look like, and neither systems implementors of languages supporting these DM's nor their users are sure of the exact meaning of the constructs, especially those in the data manipulation language (DML). Furthermore, the recent interest and importance of database mappings makes it even more urgent that DM's be given a clear, correct and complete description. As in the case of programming languages, the answer lies in formal descriptions of DM's, and their DML's, which should augment the informal discussions as final arbiters of what the model means.

Just as in programming languages where we want to describe the meaning of type declarations and groups of statements, in database management, we wish to present the semantics of schema declarations and DML's. In the case of databases the permanent nature and importance of the actual data, contrasted with the transient nature of transactions, make it even more important to express conditions on what are the valid states of the database (e.g. keys must be unique in relational theory).

In a landmark paper, Hoare and Lauer ([Hoare & Lauer73]) argued that by providing several formal descriptions of the semantics of programming languages, one may reach a wider audience of potential language users. For example, even assuming that programs map machine states to other states, one can define the meaning of a program b to be (i) the relation R_b of all initial-final state pairs (relational semantics) (ii) the relation P_b of assertion pairs (α , β) such that if b is started in a state with α holding, and b halts then β is true in the final state (partial correctness semantics; and (iii) the predicate transformer [b] ([Dijkstra76]) etc. As noted by Greif and Meyer ([Grief & Meyer79]), in each case there may be several ways of specifying the semantics: inductively, deductively, axiomatically, etc.

The multiplicity of descriptions is useful for various audiences. Thus the state view is closest to the needs of the database implementor. On the other hand, the partial correctness assertions (PCA's) provide a useful tool for proving properties of the applications programs. In the database domain this ability may play a crucial role in dealing with integrity constraints ([Hammer & McLeod75]). One of the purported advantages of centralized databases is the control over the "quality" of data and its correctness. Given constraints such as "all employees must earn no more than their managers", one could check after every database update to see if this constraint is maintained. Alternatively, if access to the data base is restricted to take place through pre-defined transactions, we can prove once and for all that each transaction maintains the integrity constraints and then be assured that the integrity assertion is valid at all times without the need for further checks. The verification of such program properties is usually done using a proof theory (e.g. [Hoare69]). However, this must be shown to be at least consistent to the usual state-change semantics in order that the proofs have any connection with the actual implemented and running system.

Although [Grief & Meyer79] indicate that the way in which the various semantics and presentations depend on each other may be quite subtle (and were omitted in [Hoare & Lauer73]), the work of [Cook78] has indicated a very nice way to show the appropriation of the deductive theory: soundness and relative completeness.

The goal of this paper is then two-fold. First, we present a data model which is the kernal of the TAXIS model [Mylopoulos, Bernstein, Wong80], but which closely resembles a number of other recently introduced models ([Smith & Smith79],[Codd79]), and we define the intended semantics through a mathematical model with well-formedness constraints expressed both directly and through axioms in an assertion language which specify conditions on "admissible" database states. Secondly, we define a DML for expressing transactions over the above data model, and specify the semantics of the language constructs in two ways: relationally and through partial correctness assertions (PCA's). For the latter purpose, we describe an extended assertion language for stating PCA's about programs and present a deductive theory of the Floyd-Hoare style for proving PCA's. Most significantly, we give the principle new steps required in showing that the proof theory is both sound and complete in the sense of Cook ([Cook78]), a step missing from many other specifications (e.g., [Gardarin & Melkanoff80]).

As a result, among others, we give the first "correct" axiomatization for For-loops, deal appropriately with attribute updates and obtain an assertion language whose proof theory can also use the aforementioned defining axioms of the data model.

We conclude by briefly comparing our work with other semantic specifications for databases with similar goals to ours and summarizing our results.

2. The TAXIS Data Model

2.1 The Schema

TAXIS is an object-oriented language for describing databases and application programs, called transactions. As a modelling tool of some slice of reality, TAXIS assumes, as do most recent semantic models, that our conceptualization of the world is populated by objects which are inter-related, and it is the goal of the database users to capture the current state of our world knowledge (i.e., what are the objects whose existence is asserted and how do they relate to each other). In TAXIS, relationships between objects will be expressed through (factual) properties which are functions (e.g., john's age is 24).

The description of the database, and its current state is made more manageable by describing first a schema to which all database states must conform. In TAXIS this is accomplished by defining classes to which objects must belong and specifying for each class its "definitional properties", i.e., the allowable properties which its instances may have, together with any restrictions which the property values must satisfy.

Note that a class, like a relation ([Codd70]), plays a dual role:

-- as a set name, it defines an extension - collection of <u>current</u> instances.

-- as a type definition mechanism, it restricts the description of its potential instances in terms of properties and property values.

TAXIS allows for the description of several types of data classes.

(a) a number of <u>basic</u> (<u>built-in</u>) <u>classes</u> such as INTEGER and STRING.

(b) <u>finitely</u> <u>defined</u> <u>classes</u>, resembling PASCAL subrange and scalar types, defined using a syntax like

<u>define</u> AGE : = {| 0::160 |} ;

or

<u>define</u> SEX : = {| male, female |} ;

<u>define</u> DIGIT : = {| '0','1',...,'9' |}

(c) <u>form</u> <u>defined</u> <u>classes</u>, obtained by "concatenating" together strings of other classes, e.g.

<u>define</u> PHONE NO. : = DIGIT \@ DIGIT \@ DIGIT \@ {|'-'|} @ DIGIT @ DIGIT @ DIGIT @ DIGIT ;

Note that in all of the above cases the set of instances of a class is fixed once and for all and the instances of the class have no properties.

(d) _variable_ _classes_, resemble relations; they describe potential instances in the way that relation schemas or PASCAL record types do, by describing the properties (attributes) applicable to its instances and classes (domains) which act as the range of property values. For example,

```
define variable-class PERSON with

     name:      PERSON-NAME;
     address:   ADDRESS-VALUE;
     age:       AGE;
     spouse:    PERSON
```

In addition, one may make further restrictions on possible property values by

-- specifying a subset of properties as _keys_ (i.e., all instances of the class must have unique combination of key-property values)

-- specifying a subset of properties as immutable (i.e., not modifiable by update operators)

-- specifying an _integrity_ _constraint_ - an arbitrary formula in the First Order Language to be defined below; For example,

```
     key (name, address)
     immutable (name, address, sex);
     !(∀x) (is(x,PERSON) ⊃ ˜(x.spouse = x))
```

TAXIS provide two special variable classes, ANY, which by definition, has as instances all instances of all other variable classes, and NONE, which has no instances. For descriptive purposes, if C is a variable class and p one of its properties with definitional property value (domain) D, then Coop can also be used to refer to D.

The instances of classes in TAXIS are called tokens (tuples in the relational model) and, as mentioned, they have factual properties and values conforming to the definitional properties provided for by the classes which they belong to. If t is a token and p a property, top is used to refer to the value of property p for t. TAXIS has special token nothing to indicate "no value" (vs unknown for missing value, see [Mylopoulos & Wong80]) and special binary predicate is which evaluates to true iff its first argument is a token instance of the second argument, which is a class. A database state can then be defined extensionally by the is predicate, by storing all current instances of classes, and the property values for all these tokens.

2.2 Well-formed Database States

The following is a list of conditions which all TAXIS database must satisfy in order to be well-formed. Like the uniqueness of keys for relational databases, these conditions are part of the definition of the TAXIS data model and hence must be captured in the semantic description.

(1) For every token t and every property p there is at most one token a such that top = a.

(2) For every property p and classes C,D such that Coop = D, it must hold that for every instance t of C, there is an object a such that top = a, and either a is <u>nothing</u> or a is an instance of D.

(3) For every property p, and tokens t and a, if top = a then there exist classes C and Coop such that t is an instance of C and a is an instance of Coop.

(4) No two instances of C may have the same combintion of values for the key properties and not all key properties can have <u>nothing</u> as value.

(5) All tokens which are instances of variable classes are also instances of ANY;

(6) NONE has no instances.

3. An FOL for Describing the Database: L_2

In order to state conditions on the data model, including integrity assertious, and for later use in the DML, we define L_2, a First Order Language (FOL) with identify by specifying, as usual, the constants, variables, function symbols, predicate symbols, terms and well-formed formulas (WFF's) of the language.

Constants and variables come in three sorts

token

-- Constants: numbers, strings, , all scalars listed in finitely-defined classes or obtained by concatenation in form-defined classes.

-- Variables: n, m, x, y

class

-- Constants: ANY, NONE, INTEGER, STRING, all class names defined;

-- Variables: X, Y, Z

property

-- Constants: all properties mentioned in variable class definitions

-- Variables: x_p , y_q

Note that one cannot name directly tokens belonging to variable classes (tuples), but one can use tuple variables x,y ranging over the class ANY (viz surrogates in [Codd79]).

Among the functions and predicates we include all the standard ones dealing with numbers and strings (e.g., +, -, /, *, mod, <, >, ≠, substr, ...). In addition, the factual property value function o maps tokens and properties to tokens, while oo, the definitional property value function, maps classes and properties to classes. Finally the predicate is relates tokens to classes, and equality = is assumed to have its standard meaning for all sorts.

The terms of the language are defined recursively in the normal way with infix notation for all functions. In addition we restrict "tuple expressions" of the form eop, when p is a property, so that in all such cases e must have the form x or xop_1, or xop_1op_2 or ... where x is a tuple variable and $p_1,p_2,\ldots$ are property names.

Atomic formulas are obtained by applying binary predicates is, =, <, > to terms, and WFF's are defined by combining atomic formulas with logical connectives $\vee$, $\wedge$, $\neg$, $\supset$, $\leftrightarrow$, and quantifiers $\forall$ and $\exists$.

Integrity constraints are then WFF's in this language L_2 and an additional well-formedness condition on a TAXIS database is that it must satisfy the constraints stated.

4. Semantics of the TAXIS Data Models

4.1 Interpretations

To give the semantics of a data model involves mapping from the syntactic objects defined in Sections 2 and 3 into a domain of mathematical objects. The tranditional way of doing this is by defining the notion of "state". In our case, we start by defining a schema model DB as a 7-tuple:

$$DB = (\hat{D},\hat{C},\hat{P},\perp,0_c,1_c,\text{p-val})$$

where

$\hat{D}$ is the domain of tokens, consisting of the union of

- $\hat{N}$ - the natural numbers
- Σ^* - the strings over alphabet Σ
- $\hat{T}$ - an infinite linearly ordered domain distinct from $\hat{N} \cup \Sigma^*$
- $\perp$ - the special object in $\hat{T}$, greater than all other elements in $\hat{T}$.

$\hat{C}$ is a domain of entities, containing two special elements $0_c,1_c$

$\hat{P}$ is a domain

p-valis a partial function from $\hat{C} \times \hat{P}$ $\hat{C}$

A <u>state</u> σ of a schema model DB is then a 3-tuple (delta,π,$\rightarrow$) where

δ is a function from variables to $\hat{D} \cup \hat{C} \cup \hat{P}$

π is a partial function $\hat{T} \times \hat{P} \rightarrow \hat{D}$

$\rightarrow$ is a relation between $\hat{D}$ and $\hat{C}$ (i.e., a subset of $\hat{D} \times \hat{C}$)

An <u>interpretation</u> I is then a mapping from a TAXIS data schema into DB states as follows:

I maps constant numbers and character strings into their normal meaning in $\hat{N}$ and Σ^*

I maps scalar constants into <u>distinct</u> elements of $\hat{N}$

I () = $\perp$

I maps classes into distinct elements of $\hat{C}$

I (ANY) = 1_c

I (NONE) = 0_c

I (oo) = p-val

I maps each property into a distinct element of $\hat{P}$

All standard numeric and string operations are mapped into the corresponding mathematical functions

"=" gets its usual interpretation

An interpretation I of the schema is then extended to an interpretation I_σ into DB state $\sigma = (\delta,\pi,\rightarrow)$ as

$I_\sigma(x) = \delta(x)$ for all variables x

$I_\sigma(\text{is}) = \rightarrow$

$I_\sigma(o) = \pi$

The interpretation is then extended to terms, formulas and WFF's of L_2 following methods of Fregean Compositionality (i.e., $I_\sigma(f(\alpha,\beta)) = I_\sigma(f)(I_\sigma(\alpha),I_\sigma(\beta))$.

4.2 Admissible Models

At this point we can complete our definition of the TAXIS data model semantics by expressing well-formedness constraints which must be satisfied by all databases. We can do this by placing direct restrictions on the "admissible states and interpretations".

For example, in addition to the obvious restrictions that numbers and character sequences be mapped in the natural way to numerals and strings and similarly for their respective operations, we must have

- in all states σ, $I_\sigma(t) \rightarrow I(\text{INTEGER})$ for any token expression t iff $I_\sigma(t) \in \hat{N}$
- in all states σ, $I_\sigma(t) \rightarrow I(\text{STRING})$ for any token expression t iff $I_\sigma(t) \in \Sigma^*$
- if C is the finitely-defined clases $\{|i::j|\}$ where i and j are numerals, then $I_\sigma(t) \rightarrow I(C)$ iff $I_\sigma(t) \in \hat{N}$, $i \leq I_\sigma(t) \leq j$.

- if C is the finitely-defined class $\{|k_1,k_2,\ldots,k_n|\}$, where k_i are scalar constants, then $I_\sigma(t) \rightarrow I(C)$ iff $I_\sigma(t) = I(k_1)$ or $I_\sigma(t) = I(k_2)$ or ... $I_\sigma(t) = I(k_n)$.

- if C is the form-defined class $C_1 @ C_2 @ \cdots @ C_n$, then $I_\sigma(t) \rightarrow I(C)$ iff $I_\sigma(t) = L_1 \,\|\, L_2 \,\|\, \cdots \,\|\, L_n$ where $L_i \rightarrow I(C_i)$

- for every t in T, $t \rightarrow 1_c$ iff $t \rightarrow d$ for some d in $\hat{C}$, and $t \rightarrow 0_c$ for no t.

The restrictions on property values from Section 2.2 can be stated as conditions on states:

for every p in $\hat{P}$ and c,d in $\hat{C}$ such that p-val(c,p) = d, and $t \in \hat{T}$, either $\pi(t,p) = \perp$ or $t \rightarrow c$ and there exists $t' \in \hat{D}$, $t' \rightarrow d$ and $\pi(t,p) = t'$;

- for every p in $\hat{P}$, t in $\hat{T}$, t' in $\hat{D}$, if $\pi(t,p) = t'$ then there exist classes c,d in $\hat{C}$ such that p-val(c,p) = d, $t \rightarrow c$ and either $t' = \perp$ or $t' \rightarrow d$.

- for every p in $\hat{P}$, $\pi(\perp,p) = \perp$;

- if D is a variable class with <u>key</u> properties $q_1,q_2,\cdots,q_r$ then for every t and t' such that $t \rightarrow I(D)$, and $t' \rightarrow I(D)$, $\pi(t,I(q_i)) \neq \pi(t',I(q_i))$ for some i and $\pi(t,I(q_i)) \neq \perp$ for some i

- if φ is a WFF which is stated as an integrity constraint, then $I_\sigma(\varphi)$ must be true.

Finally, the variable δ must map token variables into $\hat{D}$, class variables into $\hat{C}$, and property variables into $\hat{P}$.

Alternatively, one can impose these restrictions on interpretations and states indirectly by stating them as axioms in L_2 and then considering as admissible only those interpretations which make the axioms true. In fact we will give axioms <u>schemata</u> which depend in part on the actual data model being defined in order to avoid second order expressions.

T1 $\vdots$ <u>is</u>(n, INTEGER) for every number n

$\vdots$ $(\forall x)$ (<u>is</u>(x, INTEGER) $\supset x=1 \vee x=2 \vee \ldots$)

T2 $\vdots$ <u>is</u> (s, STRING) exactly for sequences of characters enclosed in quotes

T3 if $C = \{| i::j |\}$, then

$\vdots$ <u>is</u>(x, C) $\leftrightarrow$ <u>is</u>(x, INTEGER) $\wedge\ i<x \wedge x<j$

T4 if $C = \{| k_1, k_2, \cdots, k_n |\}$ then

$\vdots$ <u>is</u>(x, C) $\leftrightarrow x=k_1 \vee x=k_2 \vee \cdots \vee x = k_n$

T5 if $C = C_1 \,@\, C_2 \ldots @\, C_m$, then

$\vdots$ <u>is</u>(x, C) $\leftrightarrow$ $(\exists x_1, \cdots, x_m)$ (<u>is</u>(x_1, C_1) $\wedge \cdots \wedge$

<u>is</u>(x_m, C_m $\wedge\ x = x_1 \,\#\, x_2 \cdots \#\, x_m$)

T6 $\vdots$ $(\forall q)(\forall Y,Z)(\forall u)(Y \circ\circ q = Z \wedge$ <u>is</u>(u, Y) $\supset$

$[(\exists v)($<u>is</u>$(v,Z) \wedge u_o q = v) \vee u_o q =$ <u>nothing</u>$]$

T7 $\vdots\ (\forall q)\ (\forall u,v)\ (u_{o}q = v \supset (\exists Y,Z)$

$(Y_{oo}q = Z \wedge \underline{is}(u,Y)$

$\wedge\ (\underline{is}(v,Z) \vee v = \underline{nothing}))$ and

$\vdots\ (\forall q)(\forall u)\ (\neg(\underline{is}(u,ANY) \supset u_{o}q = \underline{nothing})$ and

$\vdots\ (\forall q)\ {}_{o}q = \underline{nothing}$

T8 $\vdots\ (\forall u)((\exists Y)\underline{is}(u,Y) \leftrightarrow \underline{is}(u,ANY))$

$\vdots\ (\forall u)(\neg\ \underline{is}(u,NONE))$

T9 if $q_1,\ldots,q_m$ are the $\underline{key}$ properties of class C then

$\vdots\ (\forall x,y)[\underline{is}(x,C) \wedge \underline{is}(y,C) \supset \neg\ (xoq_1 = yoq_1 \wedge \cdots$

$\wedge\ xoq_m = yoq_m)] \wedge \neg\ (xoq_1 = \bot \wedge xoq_2 = \bot,\ldots)$

T10 $\vdots\ \varphi$ for every integrity constraint φ

In order to be completely rigorous, one would of course have to prove that the two preceding definitions of "well-formedness" are equivalent. In the interest of brevity, this step will be omitted here. In addition to giving a second reading to our conditions, one which may be more understandable to some audiences, the axiomatic version will be useful when proving properties about the programs running on the database.

5. The TAXIS DML

One of the advantages of the TAXIS model is that it integrates transaction definition into the database design framework. This is done by defining a relatively simple DML, consisting of a number of simple statements and rules for building compound ones. In these statements, one can use expressions whose syntax is identical to that specified for L_2.

Simple Statements

1. no op -- <u>nil</u>.

2. assignment -- <variable> := <token-expression>.

3. token insertion -- <u>insert</u> x <u>in</u> C <u>with</u> $p_1:e_1,\cdots,p_n,e_n$;
   where x is a variable, C is one of the variable classes of the schema, $p_1,...,p_n$ are all its properties and $e_i,\cdots,e_n$ are token expressions. The effect of this statement is that variable x is assigned as value a new token which is inserted into classes C and ANY and for which $x_o p_i = e_i$.

4. token deletion -- <u>delete</u> it;
   where t is a tuple expression. Its effect is to remove the token referred to by t from all classes, including ANY, and set all property references to t to ⊥.

5. property update -- top := e;
   where t is a tuple expression, p a property name and e is a token expression; the effect is the obvious one of ensuring that top = e

6. object retrieval -- <u>get</u> x <u>from</u> C <u>with</u> $p_1:e_1,\ldots,p_m:e_m$;

   where x is a variable, C, one of the variable classes defined, $p_1,\ldots,p_m$ are its <u>key</u> properties and $e_1,\ldots,e_m$ are token expressions. The effect of this statement is to assign the variable as value the unique token instance of C with property values $e_1,\ldots,e_m$, if it exists, <u>nothing</u> otherwise.

In TAXIS, one can define complex statements using a number of <u>standard</u> control structures:

7. sequencing -- $S_1;S_2$.

8. conditional -- <u>if</u> e <u>then</u> S_1 <u>else</u> S_2;

   where e is (quantifier free) formula in L_2.

9. For-loop -- <u>for</u> x <u>in</u> C <u>do</u> S <u>od</u>;

   where x is a variable, C a class, and S a statement; the operations in S are executed once for every instance of C and it is assumed that S does not alter the set of instances of C, and that the final effect of the loop is independent of the order in which one stepped through the instances of C.

10. grouping -- <u>begin</u> $S_1,\ldots,S_n$ <u>end</u>.

Finally TAXIS allows assertions in L_1 to be interspersed with the statements of the transactions; if any of the assertions fails during program execution, an error is considered to have occurred.

In order to limit the size of this paper, we have chosen not to deal with variable declaration, parameter passing and exception specification in TAXIS. In any case these features are totally standard and have been extensively studied elsewhere ([London et al.78],[Ernst],[Cook78],[Levin77]).

In the remainder of the paper, we aim to give the semantics of TAXIS programs in terms of state transitions and then propose a set of axioms and rules of inference for proving assertions about programs. In order to gain confidence in the appropriateness of our rules, we will present proofs of soundness and completeness for some of the more novel constructs in the language.

6. Relational Sematics of TAXIS DML

Let DB be a schema model for a TAXIS database and let I be an admissible interpretation as defined in Section 4. Since TAXIS programs are assumed to be deterministic, then with every statement S we associate a partial function $M_I(S)$ from valid states to states which represents the sematics of that program. [1] This is done recursively by defining M_I for the primitive statements first.

Let σ be a DB-state $(\delta,\pi,\rightarrow)$ and let $\sigma' = (\delta',\pi',\rightarrow')$ be $M_I(S)\sigma$; then for

R1. $S \equiv$ nil, $\sigma' = \sigma$

R2. $S \equiv x := e$, then $\pi' = \pi$, $\rightarrow' = \rightarrow$,

$\delta' \overset{x}{===} \delta$ [2] with $\delta'(x) = I_\sigma e$

R3 $S \equiv$ insert x in C with p_i:e,

$\delta' \overset{x}{===} \delta$ [2] with $\delta'(x) = \min\{t \mid t \in \hat{T}-\{\bot\}(t,ANY)^{-}\in\rightarrow\}$ call it k;

$\rightarrow' = \rightarrow \cup \{(k,I_\sigma(C)),(k,I_c(ANY))\}$

R4 $S \equiv$ delete x where x is assumed to a tuple expression

Let $k = \delta(x)$ (hence $k \rightarrow \hat{T}$);

then $\delta' \overset{x}{===} \delta$ with $\delta'(x) = \bot$,

$\rightarrow' = \rightarrow-\{(k,c) \mid c \text{ in } C\}$

[1] If the resulting state is not valid, M(S) is assumed to be undefined at that point.

[2] The notation $f \overset{w}{==} q$ indicates that f and g are identical except possibly at argument w.

$$\pi' \overset{(k,q)}{====} \pi \text{ with } \pi'(k,q) = \perp \text{ for every } q \text{ in } \hat{P};$$

(Note that well-formedness requires that there be no k' such that $\pi(k',q) = k$ in σ).

R5. $S \equiv w \cdot p := e$

Let $k = I_\sigma(w), \bar{k} = I_\sigma(e)$

then $\delta' = \delta, \rightarrow' = \rightarrow$, and

$\pi'(k,p) = \bar{k}$ with $\pi = \pi'$ otherwise

R6 $S \equiv$ get object x from C with $p_i = e_i$

$\rightarrow' = \rightarrow$, $\pi' = \pi$ and

$\delta' \overset{}{===} \delta$ where

$$\delta'(y) = \begin{cases} k \text{ if there is } k \text{ such that } k \rightarrow I \\ \qquad \text{and } \pi(k,p_i) = I_\sigma(e_i) \\ \perp \text{ otherwise ;} \end{cases}$$

The meaning of compound statements is then defined recursively by:

R7 $M(S_1;S_2)\sigma = M(S_1) \cdot M(S_2)\sigma$

R8 M (<u>if</u> a <u>then</u> S_1 <u>else</u> S_2) σ

$$\begin{cases} M\ (S_1)\ \sigma \text{ if } I\sigma(a) \text{ is true} \\ M\ (S_2)\ \sigma \text{ if } I\sigma(a) \text{ is false} \end{cases}$$

R9 M(<u>for</u> y <u>in</u> C <u>do</u> S <u>od</u>)$\sigma = \sigma$ if there is no k in $\hat{T}$ such that $K \to I_\sigma(C)$, otherwise let $k_1, \cdots, k_n$ be <u>all</u> the elements of $\hat{T}$ in increasing order such that $k_i \to I_\sigma(C)$

Define states $\sigma_1, \cdots, \sigma_{n+1}, \bar{\sigma}_1, \cdots, \bar{\sigma}_n$ as follows:

$\sigma = \sigma'$

$\sigma_i = \bar{\sigma}_i$ with $I_{\bar{\sigma}_i}(y) = k_i$

$\sigma_{i+1} = M(S)\bar{\sigma}_i$

R10. M <u>begin</u> S_1; S_2; ...; S_n <u>end</u> σ =

$$\begin{cases} M\ (S_1) \cdot M \text{ \underline{begin} } S_2;\ \ldots\ ;\ S_n \text{ \underline{end} } \sigma \text{ if } u > 1 \\ M\ (S_1)\ \sigma \text{ if } n = 1 \end{cases}$$

Finally, assertions allow the programs to proceed only if they evaluate to true

R11 $M(!\alpha)\sigma =$

$$\begin{cases} \text{iff } I_\sigma(\alpha) \text{ is true} \\ \text{undefined otherwise} \end{cases}$$

These rules are assumed to determine the standard relational semantics of TAXIS programs, utilizing terminology of [Grief & Meyer79].

7 Partila Correctress Semantics

7.1 The Assertion lanugages L_3 and L_4

Following the tradition of Floyd-Hoare logic, we will define an assertion language for talking about properties of programs. This language has two parts : a language L_3 for making statements about states and then L_4, a language for making 'partial correctness assertions' (pcas) about programs in the traditional pre-, postcondition form.

We will extend L_2 L_3 by allowing new symbols whenever o and is appear in L_2. In both cases we find it simpler to describe the new function and predicate symbols allowed through simple grammars.

The grammar

E := o

E := E [E_1,p,E_2] for every property name p generates new function symbols which will be allowed to appear in L_3 wherever o appeared in L_2. The intended interpretation of a o [b,q,e,] p is for it to have value e if a=b and q=p, otherwise continue to be aop.

Similarly, the grammer

M := is

M := M [e : D] where e is any L_3 term and D is a class

M := M [¯e]

generates new predicate symbols which are allowed to appear in

L_3 whenever is appeared in L_2.

Formally, we extend interpretation I_σ of L_2 to interpretations of L_3 by defining I_σ (f) for f generated from E by

- if f = o then $I_\sigma (f) = \pi$
- if f = $\bar{f}$' [a,p,e] then $I_\sigma (f) = \delta$

where

$$\delta (Z_1, Z_2) = \text{if } Z_1 = I_\sigma (a) \text{ and } Z_2 = I_\sigma (p) \text{ then } I_\sigma (e) \text{ else } I_\sigma (\bar{f})$$

In a similar vein, I_σ (m) is defined for predicates in generated from M by

- if m = is then $I_\sigma (m) = I_\sigma \underline{is} = \rightarrow$
- if m = $\bar{m}$ [a:e], $I_\sigma (m) = I_\sigma (\bar{m}) \cup \{(I_\sigma(a), I_\sigma(e)), (I_\sigma(a), I_\sigma(ANY))\}$
- if m = $\bar{m}$ [¯a], $I_\sigma(m) = I_\sigma(\bar{m}) - \{(I_\sigma(a), d) \mid d \text{ in } \hat{C}\}$

The language L_4 of partial correctness assertions is then defined to contain formulae of the form P{S}Q where P and Q are assertions from L_3 and S is a TAXIS statement.

The infinitive meaning of the construct P{S}Q is tht whenever P holds in state σ and the execution S holds in state σ', then Q holds in σ'. Stated more precisely, we will say P{S}Q is valid under interpretation I into model DB:

$⁝_I$ P{S}Q iff for every DB-state σ, if I_σ (P) is true (written $⁝_I\ P_\sigma$) the $⁝_I\ Q_\sigma$ where $\sigma' = M_I(S)\sigma$, if σ' exists.

For convenience, we shall henceforth assume a fixed interpretation I and drop subscript I.

In order to prove assertions about general programs one will provide a set of axioms and rules of inference which will allow us to deduce formally ("prove") statements of the form ⁝ P{S}Q. For each primitive statement of TAXIS, we provide an axiom schema as follows:

A1 ⁝ Q { $\underline{\text{nil}}$ } Q for any predicate Q

A2 ⁝ Q ⟨ e/x ⟩ { x : = e } Q

A3 ($\exists$ y) (- (y = $\underline{\text{nothing}}$) $\underline{\text{is}}$ (y, ANY))
Q ⟨ $\underline{\text{is}}$ [y,e], o $[y,q_1,e_1] \ldots [y,q_n,e_n]$ / o, y/x⟩
$\underline{\text{insert}}$ x $\underline{\text{in}}$ C $\underline{\text{with}}$ $q_i = e_i$} Q where y does not occur in Q

A4. $Q\langle \underline{is}[\neg x]/\underline{is}, o[x,q_1,\bot]\ldots[x,q_n,\bot]/o\rangle\{\underline{delete}x\}Q$

A5. $Q\langle o[w,p,e]/o\rangle\{wop:=e\}Q$

A6. $(\exists t)(toq_i = e_i \supset Q\langle t/x\rangle) \vee$
$\neg(\exists t)(toq_i = e_i) \supset$
$Q < nothing / x >)\{\underline{get\ object}\ x\ \underline{from}\ C\ \underline{with}\ q_i : e_i\}Q$

For compound statements, we offer the following rules of inference (again schemata):

A7. $$\frac{P\{S_1\}Q, Q\{S_2\}R}{P\{S_i;S_2\}R}$$

A8. $$\frac{P\wedge a\{S_1\}Q, P\wedge}{P\{\text{if } a \text{ then } S_1 \text{ else } S_2\}Q}$$

A9. $$\frac{(\forall t)(\underline{is}(t,z)\supset \underline{is}\ (t,C))\wedge - \underline{is}(x,Z)\wedge R\{S\}R<\underline{is}[x,Z]/\underline{is}}{R<NONE/Z>\{\underline{for}\ x\ \underline{in}\ C\ \underline{do}\ S\ \underline{od}\}\ R<C/Z>}$$

where Z does not occur in S

A10. $$\frac{P\{S_i;S_2;\ldots,S_n\}Q}{P\{\text{begin}S_1;\ldots,S_n\text{end}\}Q}$$

A11. $P\{!a\}a \wedge P$ for assertion a

Finally, we add the inference rule

A12. $\dfrac{P \supset P', P'\{S\}Q', Q' \supset Q}{P\{S\}Q}$

The proof of some pca $\vdots P\{S\}Q$ will admit lines from the deductive theory of L_3 (which includes the theory of numbers and the axioms of the data model TI to TID) as well as the theory of L_4 (axioms and rules A1 to A10) and the rules of standard first order logic.

8. Soundness and Completeness of the Proof Theory

In order to have confidence in the pca derived using the rules from the previous section, we must show that they conform to the semantics of the TAXIS programming language, described in Section 6. In other words, we have to prove that the theory is sound:

$$\text{if} \;\vdots\; P\{S\}Q \text{ then } \models_I P\{S\}Q \;.$$

To do this, it is sufficient to verify all axioms and rules of inference presented. We will do so only for axiom A3 and rule A9; the other proofs are quite similar and straightforward.

Assume a fixed interpretation I for the remainder of the proof. To verify axiom A3, we must show that if we let R be

$$(\exists y)(\tilde{}(y =) \wedge \tilde{}\ \underline{is}(y,ANY) \wedge Q < y/x,$$

$$\underline{is}[y,C]/\underline{iso}[y,p_1,e_1][y,p_2,e_2]\ldots/o >)$$

then $\models_I R\{\underline{insert}\ x\ \underline{in}\ C \text{ with } p_i : e_i\}Q$. Let σ be any state where R is true and let $\sigma' = M(\underline{insert}\ x\ \underline{in}\ C\ \underline{with}\ p_i{:}e_i)\sigma$. To begin with note that in order to show that

$$I_\sigma(R<\alpha_1/\beta_1,\ldots,\alpha_n/\beta_n>)$$

implies

$I_\delta(R(\delta_1/\beta_1, \cdots, \delta_n/\beta_n>)$

where σ and τ are arbitrary states, it is sufficient to show that I_σ and I_τ agree - everywhere except at $\beta_1, \cdots, \beta_n$ and that $I_\sigma(\alpha_i) = I_\tau(\delta_i)$. In our case, by the assumption that $I_\sigma(R)$ is true, there exists token j such that $\delta_\sigma(y) = j$, $j \neq \perp$, not $j \rightarrow I(ANY))$ (and hence by the semantic restrictious in Section 4.2, not $j \rightarrow d$ for any d in C); furthermore, by definition of L_3, $I_\sigma(is[y:C]) = I_\sigma(is) \cup \{(j, I_\sigma(C)), (j, I_\sigma(ANY))\}$ and $I_\sigma(o[y, p_1, e_1] \ldots /o) = g$ where $g(z_1, z_2)$ equals $I_\sigma(o)$ unless $z_1 = j$, $z_2 = I_\sigma(p_i)$ for some i where $g(z_1, z_2)$ equals $I_\sigma(e_i)$. Therefore, defining $\sigma" = (\delta", \pi", \rightarrow")$ such that $\delta"(x) = j, \pi"(j, p_i) = I_\sigma(e_i)$ and $\rightarrow" = \cup \{(j, I_\sigma(C)), (j, I_\sigma(ANY))\}$ and $\sigma" = \sigma$ otherwise, we get (by our note above) that $I_{\sigma"}(Q<x/x, is/is, o/o>$ since $I_\sigma(Q<y/x, is[y, C]/is, o[y, p_1, e_1] \ldots /o>)$ was assumed to be true. On the other, the semantics of <u>insert</u> (R3) show σ' to be equivalent to $\sigma"$ since the replacement of j by k makes $\sigma"$ identical to σ'. But then by our definition, Q is also true in σ' since we only consider equivalent states.

Next, to verify the soundness of rule A9, let R be a predicate and let σ be a state such that R<NONE/R> is true in σ. Then, considering the definition of relational semantics R9 in Section 6, let $\tau \equiv \sigma$ except that $\delta_\tau(Z) = d, d$, a distinct value from all other classes and $\rightarrow_\tau = \rightarrow_\sigma$. But this means that Z has no instances in state τ and $I_\tau(R)$ msut be true because it has the same values for

I_σ(NONE) and $I_\tau(z)$. (If R⟨NONE/z⟩ was true in σ then R could not have had expressions of the form Zoop).

Letting $k_1,\cdots,k_n$ be the instances of C in σ, define states $\tau_1,\ldots,\tau_{n+1}$, $\bar{\tau}_1,\cdots,\bar{\tau}_n$, $\dot{\tau}_1,\cdots,\dot{\tau}_n$ according to the following diagram

$$\tau = \tau_1 \xrightarrow{y:=k_1} \bar{\tau}_1 \xrightarrow{S} \dot{\tau}_1 \xrightarrow{\text{insert } y \text{ in } Z} \tau_2 \xrightarrow{y:=k_2} \bar{\tau}_2 \xrightarrow{S}$$

$$\dot{\tau}_2 \xrightarrow{\text{insert } y \text{ in } Z} \cdots \xrightarrow{\text{insert } y \text{ in } Z} \bar{\tau}_{n+1}$$

But then the predicate $(\forall t)(\underline{is}(t,z) \supset \underline{is}(t,C)) \wedge \underline{is}(y,C) \wedge \underline{is}(Y,e) \wedge \underline{is}(y,Z)$ is also true in $\bar{\tau}_1$, s that R⟨$\underline{is}$[y,Z]/$\underline{is}$⟩ is true in

τ_1 by the premise of the inference rule. One can then repeat the argument in states τ_2, etc. until we reach τ_{n+1} where R is true and z has the same set of instances of C does in σ_{n+1} since s is not allowed to alter the set of instances of C. But the R⟨C/Z⟩ is true in σ_{n+1} since, as we noted before, R does not have expressions of the form Zooq ∎

Conversely, one would want to have some confidence that all pca's which are true of a program can be derived using the proof rules. That is we wish to show the completeness of our proof theory:

if $\vdots_I$ P{s}Q then $\vdots$ P{s}Q. But of course, the incompleteness of the proof theory of numbers assures that this cannot be the case. Instead we use

Cook's definition of relative completeness:

Let

$S_{ps}(Q,S) = \{\tau \mid$ there exists σ such that $F_I Q(\sigma)$ and $m(S)\sigma = \tau\}$. Then L_3 is expressive relative to I and L_2 if for every assertion Q and program *S*, there is an assertion $\overline{Q}$ in L_3 which is true exactly in the states in Sps(Q,S). We will denote $\overline{Q}$ by Sps(Q,S).

A proof system for TAXIS program is then complete in the sense of Cook if for every interpretation I such that L_3 is expressive, $F_I P\{S\}Q$ implies T_{r_I} $P\{S\}Q$ where T_{r_I} is the set of all assertions L in L_3 such that $F_I L$.

Considering axioms A1 to A6, note that for each primitive statement S_0 and predicate Q, the axioms prescribe a predicate which we can denote as $[S_0]Q$, such that $[S_0]Q\{S_0\}Q$. The important observation is that if σ is any state such that Q is true in $M(S_0)\sigma$ then $[S_0]Q$ must have been true in σ. But then, if R is a predicate such that $F_I R\{S_0\}Q$, then whenever R is true in σ, $[S_0]Q$ is also true in σ, i.e., $F_I R \supset [S_0]Q$. But then $R \supset [S_0]Q$ is probable since we assume an oracle above provability in L_3 and $[S_0]Q\{S_0\}Q$ by the axiom, leading to $R\{S_0\}Q$ by inference rule A11.

The inference rules are proven complete by induction on the size of the program S. We will consider only the novel rule A9, since the others have already been treated in the literature [Cook 78]. The completeness of rule A9 is of independent interest since there are a number of proposed rules for For-loop ([Hoare & Wirth?],[London 78],[Gardarin & Melkanoff]) which are demonstrably incomplete [Borgida 81].

Consider then arbitrary predicates P and Q and statement sequence S such that $\models_I P\{\underline{for}\ x\ \underline{in}\ C\ \underline{do}\ S\ \underline{od}\}Q$. In order to obtain P{for x in C do S od}Q, it is sufficient to find predicate R and variables Z,y <u>not</u> occurring free in S, P, or Q such that

(i) $\models_I (\forall t)\ (\underline{is}\ (t,z) \supset \underline{is}\ (t,C)) \wedge \underline{is}\ (y,C) \wedge \neg\underline{is}\ (y,Z)$

$\wedge\ R\{S\}R < \underline{is}\ [y,Z]/\underline{is} >$

(ii) $\models_I P \supset R < none/Z >$

(iii) $\models_I R < C/Z > \supset Q$

To this end, define R to be $Sps(P, \underline{for}\ x\ \underline{in}\ Z\ \underline{do}\ S\ \underline{od})$. Then, $\models P \supset R < NONE/Z >$ since by the summation of For-loops, (R9), $\underline{for}\ x\ \underline{in}\ NONE\ \underline{do}\ S\ \underline{od} \equiv \underline{nil}$ and $Sps(P,\underline{nil}) = P$. Also $\models R < C/Z > \supset Q$ by our assumption and definition of Sps. This leaves (i) to be proven. Let σ be any state where $(\forall t)(\underline{is}(t,Z) \supset \underline{is}(t,C)) \wedge \underline{is}(y,C) \wedge \neg\underline{is}(y,Z) \wedge R$ is true. By definition of R as strongest post-condition, there exists σ_0 such that P is true in σ_0 and $M(\underline{for}\ x\ \underline{is}\ Z\ \underline{do}\ S\ \underline{od})\sigma_0 = \sigma$ and let $\sigma' = M(S\langle y/x\rangle)\sigma$.

Finally, define σ_0^* to be identical to σ_0 except that $\rightarrow_{\sigma_0^*} = \rightarrow_{\sigma_0} \cup \{(I_{\sigma_0}(y), I(Z))\}$, and let $\sigma'' = M(\underline{for}\ x\ \underline{in}\ Z\ \underline{do}\ S\ \underline{od})\sigma_0^*$. Then, by definition of Sps, R must be true in σ_0^*. Furthermore, σ^* differs from σ' only by the fact that y is an instance of Z in σ''. This is true because of our careful restrictions on the relationship between S,C,Z, namely

(a) S is independent of Z (so it is not affected by <u>is</u> at Z)

(b) S does not alter the value of y nor the instances of C (so is (y,C) in σ_0).

But then, if R is true in σ'', then R < is[y,Z]/is > must be true in σ', concluding our proof. □

We conclude by remarking that, as elsewhere (e.g., [London 78]), we do not capture the complete aspects of implementation issues in our relational and pca semantics. In particular, constraints such as uniqueness of keys, non-dangling references, etc., are not checked explicitly in the specifications in Sections 6 and 7. There is, however, no theoretical obstacle for incorporating them in the rules.

10. Conclusions

We have presented in this paper the semantics of a kernel of the TAXIS database model and data manipulation language. This language is prototypical of the more recent data models ([Codd 80],[Chen 76]) and in addition integrates into one coherent framework the schema description, the DML and integrity assertions. We presented the "relational" semantics of the TAXIS model, by describing a mathematical model for its schema, including constraints on admissible database states, and by specifying the state transition semantics of the DML constructs. Based on this mathematical model of the semantics, we built up an exiomatization and proof theory with integrity partial correctness assertions about TAXIS programs with axioms about the schema and the proof theory of the First Order Predicate Calculus with identity for integers and strings. By proving the soundness and relative completeness of our proof theory, we have gained some confidence that this specification is not inconsistent with the relational semantics and that it provides a useful tool for proving properties of programs of a DML.

By omitting this last step, previous exiomatizations have fallen short of the derived standards. For example, the axiomatization of a DML in [Gardarin & Melkanoff 79], intended for a similar domain, is not complete and possibly not sound. Their rule of inference for assignment,

$$\text{DO:} \quad Q < f/x > \{\xi = f\}Q \ ,$$

is either incomplete, by not dealing at all with triple attribute value updates such as Z_0age := 20, or it is not sound since the assertion

$$\neg(y_0\text{age} = 20)\{Z_0\text{spouse}_0\text{age} := 20\}\neg(y_0\text{age} = 20) \ ,$$

provable by the rule DO, is not true when Z_0spouse = y. Also, the inference rule for For-loops

$$\frac{P_1 \wedge B\{S\}P_1}{P_1\{\underline{\text{for}}\ a\ \underline{\text{st}}\ B\ \underline{\text{do}}\ S\}P_1} \qquad \text{(D5)}$$

is incomplete because although

$$(\underline{\text{true}})\{\underline{\text{for}}\ x\ \underline{\text{st}}\ -\ \underline{\text{do}}\ x\ \text{age} := 1\}(\forall_y)(y_0\text{age} \neq 0) \qquad (*)$$

is clearly valid, the only way to prove it would be to find P_1 such that, among others, $\underline{\text{true}} \supset P_1$ and $P_1 \supset (\forall_y)(y_0\text{age} \neq 0)$; but true $\supset P_1$ means P_1 is a tautology and yet, under any sensible interpretation $\underline{\text{true}} \supset (\forall_y)(y_0\text{age} \neq 0$, thus showing that no such P_1 exists and that * is not provable.

In conclusion, we feel that theoretical exercise such as that carried out above is an important aspect of the definition of data models and their manipulation languages, one which provides indispensable support for real-life applications by presenting consistent and complementing views of the model aimed at diverse audiences such as systems designers, applications programmers, and verifiers.

Acknowledgments

We are grateful to Steven Cook, Charles Rackoff, and John Mylopoulos for revealing discussions and important pointers to the literature of program verification.

Special thanks to Carole Agazzi for her excellent typing.

References

Borgida, A., "Variations on the theme of Floyd-Hoare axioms of For-loops", unpublished manuscript, DCS, University of Toronto, February 1981.

Casanova, M. A., P. A. Bernstein, "A Formal System for Reasoning about Programs Accessing a Relational Database", _ACM Transactions on Programming Languages and Systems_, 2,3 (July 1980).

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", _CACM_ 13,6 (June 1970).

Codd, E. F., "Extending the Relational Model of Data to Capture More Meaning", _ACM Transactions on Database Systems_, 4,4 (December 1979).

Cook, S. A., "Soundness and Completeness of an Axiom System for Program Verification", _SIAM J. Comput._, 7,1 (February 1978).

Earnst

Gardarin, G., M. Melkanoff, "Proving Consistency of Database Transactions", Proc. 1979 VLDB (October 1979).

Greif, I., A. R. Meyer, "Specifying the Semantics of While-Programs: A Tutorial and Critique of a Paper by Hoare and Lauer", M.I.T. Laboratory of Computer Science, TM-130 (April 1979).

Hammer, M., D. McLeod, "Semantic Integrity in a Relational Database System", Proc. 1975 VLDB (August 1975).

Hoare, C. A. R., "Axiomatic Basis for Computer Programming", 12,10 (October 1969).

Hoare, C. A. R., P. Lauer, "Consistent and Complementary Formal Theories of the Semantics of Programming Languages", Acta Informatica, 3, pp. 135-155, 1973.

Hoare, C. A. R., N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL", Acta Informatica, 2,4, 1973.

Levin, R., Program Structure for Exceptional Condition Handling, Ph.D. Thesis, DCS, Carnegie-Mellon University, 1977.

London, R., et al., "Proof Rules for the Programming Language EUCLID", Acta Informatica, 10,1, 1981.

Mylopoulos, J., Bernstein, P. A., H. K. T. Wong, "A Language Feature for the Design of Interactive Information Systems", ACM Transaction on Database Systems, 5,3, 1980.

Mylopoulos, J., H. K. T. Wong, "Some Features of the TAXIS Data Model", Proc. 1980 VLDB, Montreal, 1980.

Oppen, D., Program Verification and Logic, Ph.D. Thesis, DCS, University of Toronto, 1975.

Smith, J. M., D. C. P. Smith, "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems, 2,4, 1977.

Tsichritzis, D., F. Lochorsky, Data Models, Academic Press, to appear in 1981.