ium on Programming. Proceedings.
M. Paul, VI, 262 pages. 1984.

Computer Integrated Manufacturing.
embold. XVI, 528 pages. 1984.

Shift Registers. II, 1–2, 145 pages.

Conference on Automated Deduction.
pages. 1984.

Decision Problems and Complexity.
y E. Börger, G. Hasenjaeger and
14.

s and Programming. Proceedings,
VIII, 527 pages. 1984.

ypes. Proceedings, 1984. Edited by
G. Plotkin. VI, 391 pages. 1984.

ceedings, 1984. Edited by J. Fitch.

s and Boolean Matrix Factorization,

ations of Computer Science 1984.
M.P. Chytil and V. Koubek. XI, 581

ages and Their Definition. Edited by
1984.

Ergonomics – Mind and Computers.
S.C. van der Veer, M.J. Tauber, T.R.G.
ages. 1984.

Matrices Faster. XI, 212 pages. 1984.

terfaces. Proceedings, 1983. Edited
1984.

ftware Technology and Theoretical
gs, 1984. Edited by M. Joseph and
ges. 1984.

al Symposium on Theoretical Aspects
dings, 1985. Edited by K. Mehlhorn.

CIP. Volume I: The Wide Spectrum
anguage Group. XI, 275 pages. 1985.

An Advanced Course. Proceedings,
J. Mariani and D. Shepherd. VIII, 497

ations of Software Development. Pro-
Colloquium on Trees in Algebra and
ted by H. Ehrig, C. Floyd, M. Nivat and
85.

Software Development. Proceedings,
n Software Engineering (CSE). Edited
and J. Thatcher. XIV, 455 pages. 1985.

ime Series Package (TSPACK). III,

Nets 1984. Edited by G. Rozenberg
rich and G. Roucairol. VII, 467 pages.

gon. XI, 376 pages. 1985.

art, G. Hommel, L. Lamport, B. Liskov,
ider, Distributed Systems. Edited by
573 pages. 1985.

of Verification Techniques for Parallel
5.

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

239

Mathematical Foundations of Programming Semantics

International Conference
Manhattan, Kansas, April 11–12, 1985
Proceedings

Edited by Austin Melton

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

ON DENOTATIONAL SEMANTICS OF DATA BASES

Naphtali Rishe

Department of Computer Science
University of California
Santa Barbara, Ca 93106

ABSTRACT

A method of denotational formalization of data bases, of data base management systems, and of related structures is proposed, aiming to improve their understanding, specification and rigorous investigation. The method provides a uniform treatment of different information layers: from instantaneous data bases (the first layer), via schemata and integrity rules, to classes of data base models. It unifies in one mathematical notion the apparently different notions of the semantics of stored data, semantics of data base processes, conceptual semantics of data bases, integrity semantics of data bases and denotational semantics of languages. The unification is based on hierarchies of domains of continuous mappings between different representations of information (from "less semantic" representations into "more semantic" ones).

1. A Hierarchy of Information Levels

In this study of data bases the term ***information*** means somehow closed or complete knowledge — with respect to certain criteria that are informally described in this section. This term may be used in plural, *e.g.* "two instantaneous data bases may represent two informations."

In order to treat data base semantics, we have to distinguish between at least three levels of information related to a data base system. (These levels should not be confused with ANSI-SPARC layers — internal, conceptual, and external — which are in fact orthogonal to the classification described in this section.) The following is a preliminary discussion, not aiming yet to define formal semantics. Neither information representations nor their semantics are studied in this section.

It is customary to classify data base information according to levels of its descriptivity. Usually, three levels are distinguished. These three levels are:

1) Inf^1—the information represented by an instantaneous data base (one Inf^1 object is the information represented by a whole instantaneous data base, *e.g.* a collection of tables).

2) Inf^2—the information about the common properties of Inf^1 information which can be represented at any instant of time by a given data base (provided the properties and the purpose of the data base are kept constant). A part of an Inf^2 information is represented by the schema of a data base.

   An Inf^2 information can contain or impose certain laws on a data base, some of them are of the following types:

   - *integrity laws*—specifying principally what are the valid states of this data base;
   - *inference laws*—specifying how from information entered to the data base other information can be deduced (by the DBMS or elsewhere);
   - *laws of operations*—specifying what operations (including atomic, complex, queries, updates or whole processes, sessions, etc.) are permissible on the data base and what are their results depending on the states of the data base. A law of operations may be a consequence of the aforementioned laws.

   These laws and other parts of an Inf^2 information can be accumulated in a data base schema, *exit routines*, basic application programs, etc.

3) Inf^3—the information describing all possible Inf^2 informations which a data base can posses under a given DBMS. An Inf^3 information can be considered as accumulated (at least partly) in the code of a DBMS.

   More important parts of an Inf^3 are syntax and meaning of data base languages, including data manipulation languages (programming), a data definition language in which schemata and some laws are defined, a query language, etc.

Inf^2 level requires further discussion. A part of an Inf^2 information can be represented by a schema of the data base according to a certain data base model, *e.g.* the Relational Model.

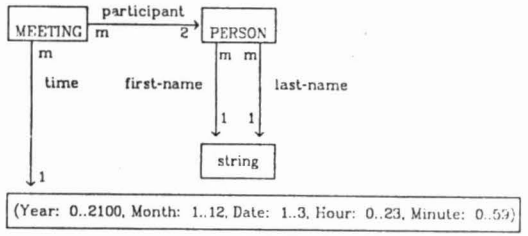
Example 1.1.

The following is a relational schema of a data base recording meetings between pairs of participants (having *Id*entification numbers, first names and surnames):

(*MEETINGS*(*Id1, Id2, Time*: integers), *PEOPLE*(*Id*: integer; *F-name, S-name*: strings))

Another part can be represented by an additional "semantic schema" (as it is called by some authors.)

Example 1.2.



The part of information which is represented by a "semantic" schema is syntactically formalized in some data base systems. In other systems this is "just" known to the community of users of the data base.

Example 1.3.

An integrity law for the sample data base consists of two parts:

a) the part in responsibility of the DBMS, which should reject all the data base updates violating any of these constraints:
   - *F-name* and *S-name* of PEOPLE depend functionally on *Id*;
   - the integers and the strings obey implementational restrictions (width of fields, etc.);
   - the instantaneous data base matches the schema.

b) though it is desirable to have the DBMS responsible for most of the constraints, the following constraints, which form a part of the integrity law, are left in most systems to a user's responsibility and are implemented by *exit routines*, by preprocessing, or by outside manual sorting of the input.
   - every participant's *Id* in MEETINGS must be in PEOPLE;
   - the representations of time are legal according to the pattern "yymmddhhmm";
   - the projection of MEETINGS to the columns (*Id1, Id2*) is irreflexive.

Example 1.4.

An inference law for the sample database, which (the law) is not supported by an unsophisticated DBMS, but is supported by application software or is just understood by the user, can be represented by the following statement:

"if A met B at t then B met A at t too".

Example 1.5.

The following part of an update law would usually be in the responsibility of application software:

a) when (*id1*, *id2*, *t*) is deleted from MEETINGS, (*id2*, *id1*, *t*) must be deleted too;

b) when (*id1*, *id2*, *t*) is inserted into MEETINGS, the presence of *id1* and *id2* in PEOPLE must be verified.

The five examples given above are parts of one Inf^2 information of the sample data base.

For some purpose, more than three levels are considered. *E.g.*, the above Inf^1 level can be subdivided into two: the lower corresponding to less *general/stable* data in the data base and the higher corresponding to more *general/stable* data; the above Inf^2 level can be split into a lower level corresponding to *subschemata* or *user-views* and a higher level corresponding to schemata; the above Inf^3 level can be split into a lower level corresponding to versions of DBMS, an intermediate level corresponding to principal DBMS, and a higher level corresponding to principal models (such as hierarchic, network, relational, binary, etc.).

The user of the proposed approach shall identify all the levels and sublevels of interest and renumber them consecutively receiving a total of n levels. For many applications n=3. For the sake of generalization, in the following sections I assume that n levels must be considered.

For any k from 1 to $n-1$, any Inf^{k+1} *information* describes a whole range of Inf^k "*informations*". At the highest level n considered, only one specific *information* is of interest. Otherwise, the user should have identified a higher level of information $(n+1)$, the only *information* of which is the list of all relevant *informations* of the Inf^n level.

2. Existing Meanings of the Term "DB Semantics" and Approaches to Its Formalization

The term "data base semantics" is widely used with different meanings. Among them: operational or denotational semantics of data base operations, logical constraints, information implication rules and user-world orientation of data base models.

Several meanings of the term "data base semantics" and approaches to its formalization are surveyed in the following:

[Earley-72] defines the semantics of a data structure as the "abstract properties it has with respect to access, possible change of structure, relationship between data items, etc."

Most authors, *e.g.* [Schmid-75], regard data base semantics as a system of constraints on information that can be stored in the data base.

[Cadiou-76] adds to this a collection of time-invariant properties of a data base.

[Weber-76] defines data semantics as a description of the information types, which should be represented in a data base for a certain purpose.

According to [Falkenberg-78], "the semantics of an application-specific universe of discourse is determined by the set of elementary facts and by the set of associated semantic rules. These semantic rules include type definitions, rules governing cardinalities, rules governing dependencies between sets, etc., and can be used for consistency checks or for deduction purposes."

Some aspects of data base semantics were formalized, at least partially,

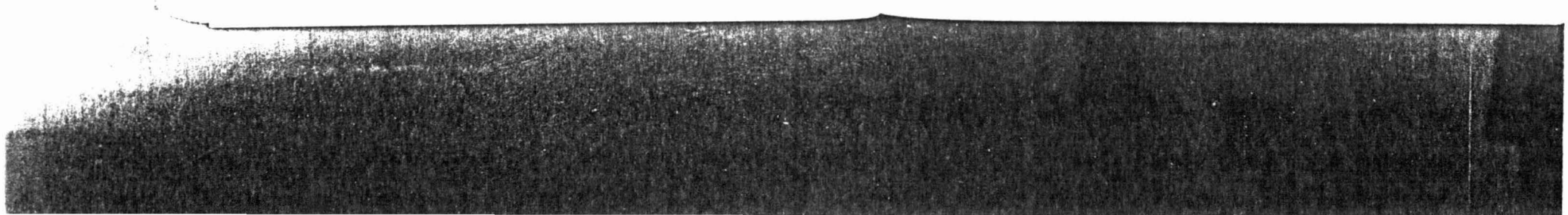
- in [Weber-76], who defined a formal (non-semantical) model of constraints and assigned to it semantics in the Operational style;
- in [Biller-76], who defined denotational semantics of data base schemata, mapping them to sets of world states, and of a data base manipulation language, mapping its programs to pairs of world states;
- in [Bjorner-80], who defined mappings from data base operations and abstract states of the data base to its new states and output;
- in [Neuhold and Olnhoff -81], who mapped the commands of a relational data manipulation language to transitions between virtual states of external and conceptual data base and to transitions between physical states of an internal data base in ANSI-SPARC three-layer data base architecture;
- in [Zaniolo-84] and [Vassiliou-79], who formalized some aspects of null-values in relational data bases;
- in [Clifford and Warren - 83], who formalized the time scale in data bases.

Recently, standardization was attempted as to the formal semantics of data base operations for the relational (table-oriented) data base model: [Brodie and Schmidt-82], [Date-82], [Hardgrave-82].

In the proposed approach an attempt is made, *inter alia*, to unify the existing meanings of "data base semantics". DB semantics is viewed as transformations of some representations of information systems (inter alia instantaneous data bases, schemata, data base management systems, data base models) into other representations thereof which are chosen (for a given user in a given environment) to be more comprehensive, more self-explanatory, more explicit, and less dependent on implementation or on linguistical or representational conventions than the former representations.

In most cases an information system of a given high level in a hierarchy of information systems will be thus mapped to a mathematical object which explicitly shows:

- how such semantics is assigned to all relevant information systems of the lower levels;



- which of these systems are not integral (detectably erroneous by the manipulating software with respect to integrity constraints, or undetectably erroneous);
- how these lower-level systems are made more explicit (due to inference rules, etc.);
- and what is the denotational semantics of manipulation languages and processes applicable to representations of information systems of the lower level.

3. Some Objectives of Defining Formal Semantics of Data Bases and DBMS

The design of the proposed approach was guided by the objectives of defining formal denotational data base semantics which are discussed in this section:

1) To provide meaningful formal description of
   - DBMS and its DDL, DML, and QUERY languages;
   - the *schema* and the administrating software complex (including exit routines, etc.) of a data base;
   - the *subschema/user-view* and the software complex of an application;
   - instantaneous data bases;
   - etc. for other levels of data base information.

   a) Such an exact specification is needed by users when certain properties are only vaguely described in manuals.

   b) It is desirable for the implementor to get an exact specification of the properties that his software should possess. After the software is programmed, its correctness with respect to the specification can be proved by the methods of programming language semantics. This is true both for the programming of a DBMS and for programming in other levels: application, exit routines, etc.

   General Examples

   1. Such a description of a DBMS should specify *inter alia* the following:
      a) what are the supportable *instantaneous data bases* and what are their limitations with respect to the widths of the fields, number of relations, etc.;
      b) what are the supportable schemata;
      c) for every supportable schema what instantaneous data bases may exist under the schema;
      d) what application software is permitted;

      e) what are the syntactically correct programs in the *Data Manipulation Language* (this DML is interpreted by the DBMS);
      f) for every given schema, DML program and *instantaneous data base*, will the program terminate, and if so what changes will it produce in the data base; what happens if several DML programs are run in parallel by several users;
      g) as above for the *Data Definition Language* with respect to alterations in schemata;
      h) as above for query languages, etc.
   2. Such a description of an *instantaneous data base* can imply some properties of an *instantaneous universe of discourse*, i.e. the relevant part of the real world at a given instant of time.

2) To study properties of constructions in different data base levels, and to have a formal basis to prove claims about these properties.

Example 3.1.

For the sample data base of the previous section we may wish to prove that there will never be one *Id* in two rows of the PEOPLE table. To prove this we need to know the semantics of the schema, *etc.* (an Inf^2) *or* the semantics of the DBMS (an Inf^3).

Example 3.2.

For the inference and integrity laws of examples of the previous section we may wish to prove that they are implementable by application software or that it does not matter what is done first: a deduction by the inference law and *then* a verification of the integrity, *or vice versa*.

3) To achieve a degree of automatization in constructing (or programming, generating, etc.) of
   - an instantaneous data base (or a part thereof), when the meaning of the information which should be represented in it is given;
   - a data base schema, a subschema and (maybe as a utopia) data base administrating and application software, according to a specification of the needed properties;
   - (maybe as a utopia) DBMS, according to requirements from it;
   - etc.

It is a long way to go from the proposals of this work to the realization of automatization, but the aim is to exploit existing and future techniques of automatic generation of programs and data structures starting from denotational specifications of these constructions or from their formalized properties which are equivalent to their full or non-full

(non-deterministic) specifications.

4) To provide a means for verification of certain constructions (such as schemata, syntactic specifications or integrity laws, system parameters, etc.) against the needed properties of the behavior of a data base. Unlike par. (2) above, here we start with denotational descriptions and produce a syntactic construction which is proved then to be correct.

Example 3.3.

Given a set of the possible histories of meetings between persons of a given potential population, and

given a formal definition of what should happen if a new person or a new meeting is reported of if wrong data (loaded earlier by mistake) is corrected, and (maybe) should the meeting or the person's data be reflected in the data base,

a schema can be produced, with its denotational semantics found and proved to match or to be equal to the definition of the needs or to imply that definition.

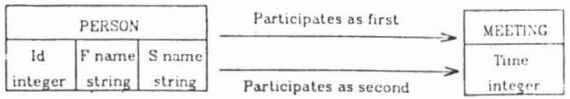
This can also be used to prove the equivalence of constructions with respect to certain criteria of their behavior or reflection of the real world. Such a proof is needed *e.g.* if the construction is technically refined or if representations need be translated from one model, *e.g. relational*, to another, *e.g. network*.

Example 3.4.

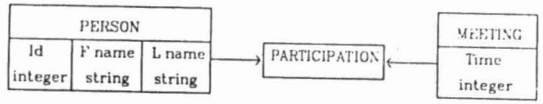
The relational schema
MEETINGS [*Id1, Id2, Time*: integers;
F-name-1, S-name-1, F-name-2, S-name-2: strings]
plus the integrity constraint:
F-name-1 and *S-name-1* functionally depend on *Id1*
F-name-2 and *S-name-2* functionally depend on *Id2*
is equivalent to the schema
(*MEETINGS*(*Id1, Id2, Time*: integers), *PEOPLE*(*Id*: integer;
F-name, S-name: strings))
plus the integrity constraint:
F-name and *S-name* functionally depend on *Id*
provided the DBMS permits "*unknown*" values in fields.

Example 3.5.

The schemata + the constraints of the previous example are equivalent to the following network schema:



And are not equivalent to the following network schema (whose application world is closer to the reality):



5) To study desired properties of the design of DDL, DML, and other related languages. This is parallel to one of the most important present-day objectives of the study of formal semantics of programming languages. Desired properties of Query languages include most of those of programming languages ("generality", "purity", "independency on representational or implementational aspects", etc.) as well as *approximation of end-user's concepts and his reasoning.*

6) When desired properties of data base behavior cannot be satisfied (implemented) by any construction (or software), to find the constructions (or software) which *approximately* (as must be defined) satisfy the properties. Among these, a best selection (with respect to certain criteria) can be chosen.

Example 3.6.

A relational DBMS is needed to support tables containing real numbers in range (0..1). This is not implementable. An approximation would be a DBMS supporting rational numbers up to a fixed number of digits.

Example 3.7.

The relevant world to be represented in a data base is too large. At some stage a part of the information has been loaded. The *instantaneous data base* at this stage can already be used for queries, etc., though it is only an approximation of the needed one.

Example 3.8.

An integrity law needs to be imposed on the data base, but its holding for some *instantaneous data bases* cannot be algorithmically checked. An implementable approximation thereof can be a law distinguishing between *integrity*, *non-integrity*, and *undefined-integrity*. This means that an implementation may loop infinitely when treating instantaneous data bases whose *integrity* is undefined.

4. Representations of Information and Data Semantics

More semantic and *less semantic* representations of information are distinguished here comparatively and subjectively according to the needs of a user of the approach and his comprehension of representations. Among two given representations of the same information usually the more comprehensive of them—more explicit and less dependent on higher-level information—will be considered by him as more semantic than the other.

Data semantics functions are defined in this section. They map subjectively "*less semantic*" representations to "*more semantic*" ones; identify erroneous representations (i.e. non-integral) which ought to be revealed by managing software; isolate erroneous representations revealing which the managing software would loop. The functions are defined to possess properties which are essential for their implementability either in a computer or by mental or manual procedures.

4.1. Domains of representations

Let us consider now *representations* of *informations* of some Inf^k level ($0<k<n$). Any one Inf^k information can be represented in different ways: implicit, explicit, computer-oriented, end-user-oriented, db-specialist-oriented, denoted by known terms, mathematically abstracted, *denotatively semanticized*, etc. Comprehending and interpreting any representation usually necessitates knowledge of an Inf^{k+1} information. When two representations of the same information are considered by a solver of a problem (a user of this approach), from the point of view of his purposes and his exploitation and perception of them, one of them can be more comprehensive, clear and usable (subjectively for the user), and its perception can be less dependent on a deep knowledge of the Inf^{k+1}, than that of the other. In this case we would call the former representation [*more*] *semantic*, and the latter *less semantic*. The proposed distinction between *less* and *more semantic* representation is comparative and not absolute. A chain of representations, from a *non-semantic* to a *very semantic* can be considered.

Example 4.1.1.

A certain Inf^2 can be represented by a schema. This representation is *less semantic* relatively to the *more semantic* representation as a set of all the possible states of the given data base. Yet, the latter representation is less semantic than a complete description of possible behavior of this data base. (One can learn of this behavior from a representation of the Inf^2 and the known Inf^3.)

Example 4.1.2.

The Inf^3 information accumulated in the software of a certain DBMS, can be represented by the string 'System-R Version 1 Release 1', which is perceivable with the use of the Inf^4, which, *inter alia*, relates all identifications of DBMS-s to their full descriptions. Another, *more semantic*, representation of the same Inf^3 is a user manual for this DBMS (for the perception of which the user "less needs to consult" the Inf^4). Actually, the latter representation can be inadequate for many purposes of semantization. A better representation will be discussed later.

When a collection of Inf^k *informations* is of interest, for every one of which there is a *more semantic* representation and a *less semantic* representation, two domains[1] can be considered, one containing (*inter alia*) the *less semantic* representations, and the other—the *more semantic* representations. Let us denote these domains $<Synt\ Inf^k>$ and $<Sem\ Inf^k>$ respectively.

4.2. Data semantics functions

Let x be a given Inf^{k+1} information, and let there be two given domains of representations of Inf^k informations, one of them chosen to contain the less-semantic representations and the other — the more-semantic ones, called $<Synt\ Inf^k>$ and $<Sem\ Inf^k>$ respectively. From x one can learn, *inter alia*, the following:

a) what *Synt* Inf^k-s (i.e. elements of $<Synt\ Inf^k>$) represent Inf^k permissible according to x;

b) what *Sem* Inf^k-s represent Inf^k informations permissible (alternatively, implementable by a *Synt* Inf^k or representable by a *Synt* Inf^k) according to x;

c) what is the correspondence between the valid elements of $<Synt\ Inf^k>$ and the valid elements of $<Sem\ Inf^k>$.

This knowledge is expressed in a *data semantics function*[2] which maps every *Synt* Inf^k representation into its data semantics, i.e. the corresponding *Sem* Inf^k representation, if the latter exists.

This function can be just a *partial function* from $<Synt\ Inf^k>$ to $<Sem\ Inf^k>$. However, we shall revise the form of these functions and the definition of domains in order to

- eliminate meaningless functions, i.e. functions which cannot potentially be data semantic functions;
- provide means for achievement of the aforementioned purpose of approximations;
- be compatible with methods of mathematical semantics of programming languages and provide means of specification and verification of software related to a data

[1] for the meanwhile, a domain is just a mathematical set, but later we shall require equipping the set with some orders.

[2] The term *data semantics* is used in order to distinguish this from the later discussed semantics of operations.

base system (*e.g.* DBMS or applications);

- simplify treatment of invalid representations;
- distinguish between different kinds of invalidity of a syntactic representation: the detectable by the DBMS and the non-detectable;
- be able to use known results on least fixed points,
- be able to treat errors, loops, routines, etc. in DML, DDL, *exit* procedures, etc.

We now assume that any considered domain is a set with a *complete partial order*[3] relating *less defined* to *better* defined elements, having the minimum element called $\perp$ or *undefined*, and having a special element called *meaningless*, or *ERROR*. Any simple set that needs to be considered is convertible to a domain satisfying the above by adding to it two special elements: $\perp$ and *ERROR*, and the *flat order*[4].

A data semantics function which was defined as partial can be extended so that those *less semantic* representations which were not mapped to *more semantic* ones will now be mapped to one of the special elements:

- *ERROR*, meaning usually that the *less semantic* representation was illegal and the managing process can detect its illegality,
- or $\perp$, meaning usually that the managing process would loop infinitely trying to check the validity of the invalid *less semantic* representation.

Every *semantic function* of interest should propagate $\perp$ and *ERROR*.

Example 4.2.1.

Consider a chain of three domains: a *non-semantic*, an *intermediate*, and a *semantic* domain, and two functions between them: f_1 and f_2 such that $f \equiv (f_1 \circ f_2)$ is a *semantic function* from the *non-semantic* to the *very semantic* domain.

If $f_1(a) \equiv \perp$ or $f_1(a) \equiv ERROR$ then a is *illegal*. Thus $f(a)$ must be respectively $\perp$ or *ERROR*. Thus $f_2(\perp) \equiv \perp$ and $f_2(ERROR) \equiv ERROR$. Thus f_2 propagates $\perp$ and *ERROR*.

A generalization of propagating $\perp$ is being *monotonic*. f is monotonic if whenever x is less defined than or equal to y, $f(x)$ is less defined than or equal to $f(y)$.

Every semantic function of interest must be *continuous* because otherwise it cannot be "implemented" by computer software or by an outside process of reasoning. This *continuity* property will be exploited later, *inter alia* for finding *least fixed points* of equations and for reasoning about *computability*.

---

[3] a *partial order* is called *complete* if for every subset of its domain there is an infimum, and for every increasing sequence in the domain has a supremum.

[4] The flat order over a domain D with minimum $\perp$ is $\{(x, y) \in D^2 \mid x \equiv \perp \bigvee x \equiv y\}$.

Unlike the above example, many domains of interest on the Inf^1 level are *flat* and for them "continuity" of a function is equivalent to its being *monotonic*. Even though, the proposed later domains for higher levels, *e.g.* Inf^2, are constructed using domains of functions over lower domains (with *pointwise* order between the functions), and they are no longer *flat*.

A reason for requesting non-flat order already on Inf^1 level can be user's need to investigate data bases from the point of view of *open-world* assumption as explained in the following.

The owner of a data base can assume either that the world is "closed" or that it is "open".

According to *closed-world* assumption every legal instantaneous data base must reflect all of the information existing in a given instant of time in the relevant (for this data base) part of the world.

According to the *open-world* assumption an instantaneous data base represents just a fraction of the relevant instantaneous information, the fraction that has been loaded into the data base: more of the relevant information possibly could have been loaded but has not been. Thus assertions that cannot be proved "true" according to the instantaneous data base can still be "true" in the reality of that time. So some instantaneous data bases can be better approximations of the reality than the other.

According to *closed-world* assumption different instantaneous data bases are not comparable. For them one would define a *flat order*.

If the user exploits the *open-world* assumption then I suggest enriching the order with the following relationships: i_1 is less defined or equal than i_2 if every simple fact reflected in i_1 is reflected also in i_2. To insure *completeness* of the order, the set of all possible simple representable facts is restricted to be finite ("finite-universe assumption").

Example 4.2.2.

Consider the sample relational data base of meetings. An instantaneous data base $(meetings-table_1, people-table_1)$ is less defined or equal than $(meetings-table_2, people-table_2)$ if the first tables are respectively included in the second ones. It is assumed that there is a given finite population and a finite time-scale. If "*unknown*" is permitted as a value in fields, then the order is enriched even more.

More generally, a data base can be *partially−open*. In this case the schema should define what types of information (relations and categories in the binary model) are *open*, and what are *closed*. An instantaneous data base would consist then of two parts: $idb = (idb_{open}, idb_{closed})$, and the order would be:

idb1 is less defined or equal to *idb2* iff $idb1_{closed} = idb2_{closed} \bigwedge idb1_{open} \subseteq idb2_{open}$.

We now define the domain containing all the possible *Data Semantics Functions* (corresponding to different Inf^{k+1}-s) from $<Synt\ Inf^k>$ to $<Sem\ Inf^k>$ as the domain of all

the *continuous* functions between these domains[5]:

$$<\textit{Data Semantics Functions (from } <\textit{Synt } \textit{Inf}^k> <\textit{Sem } \textit{Inf}^k>)> \\ = \quad [<\textit{Synt } \textit{Inf}^k> \rightarrow <\textit{Sem } \textit{Inf}^k>]$$

If an $\textit{Inf}^{k+1}$ information $\textit{inf}_1^{k+1}$ yields a data semantic function which maps a representation $s \in <\textit{Synt } \textit{Inf}^k>$ into $\perp$ or *ERROR*, then s is assumed to be illegal according to $\textit{inf}_1^{k+1}$. The difference between the illegal representations is in the ability of the system to detect the invalidity. Usually *ERROR* would denote a detectable invalidity, while $\perp$—the one which would cause the detecting software to loop infinitely.

5. Semantics of Data Base Operations

Defined here *semantics of operations* resembles Bjorner's data base semantics ([Bjorner-80]) but in my approach it is only *auxiliary semantics of operations* (a minor part of data base semantics), and the principal semantization is in *data semantics*.

Operations on $\textit{Inf}^1$ are queries, updates, etc. Operations on $\textit{Inf}^2$ are *e.g.* data definition (DDL) programs, in which schemata and laws are defined and updated.

An $\textit{Inf}^k$ level operations' semantics is defined as a function which maps every $\textit{Inf}^k$ operation together with its external input to the corresponding *continuous* transformation from old states to new states and output. (These states are *less-semantic* representations of $\textit{Inf}^k$ level information.)

Let $<\textit{Operations on } \textit{Inf}^k>$ be a *syntactic* domain of operations that are of interest and can be candidates to be executed (unless forbidden) on the k-th level in a data base system. If there are no executable operations on this level that are of interest, or if there are no executable operations at all on this level (for the DBMS level and higher), then the domain is empty. Operations can be primitive, complex, they can be whole programs, terminal sessions, parallel processes, etc. *E.g.*, for the instantaneous data base level, the syntactic domain can be equal to a data manipulation language (DML), and for the db/schema level it can be equal to a DDL.

We regard external input to an operation as an integral part of it. *E.g.*, the *query schema* like "get a name of a man from the terminal and fetch the times of his meetings from the data base" stands for the set of operations each of which is a pair composed of this

[5] This domain of functions has the point-wise *C.P.O.*, meaning that one semantics function is *less defined* than another or *approximates* it. $\text{POINT-WISE-ORDER}_{order} \equiv$
$\{(f_1, f_2) \in (dom(order) \rightarrow dom(order)) \mid \forall x \in dom(order):\ f_1(x)\ order\ f_2(x)\}$

scheme and the name of a man.

Let $<\textit{Output}^k>$ be a domain of external outputs from the operations of $<\textit{Operations on } \textit{Inf}^k>$. In many cases this can be the domain of strings over some alphabet, plus the error elements. For a given $\textit{Inf}^{k+1}$ it is known for every *operation* $\in <\textit{Operations on } \textit{Inf}^k>$, whether it is legal, and if so, how it would change the k-th level of the data base and what external output it would produce depending on the previous status of the data base. Thus, from the given information of $(k+1)$-th level, we can extract an *operations semantics function*:

$$[<\textit{Operations on } \textit{Inf}^k> \rightarrow \\ [<\textit{Representations of } \textit{Inf}^k> \rightarrow \\ <\textit{Representations of } \textit{Inf}^k> \times <\textit{Output}^k>]]$$

We have to choose which domain, $<\textit{Representations of } \textit{Inf}^k>$, is suitable for the above transformation. It could be the domain of the *more semantic* representations or the domain of the *less semantic* representations. As we have *data semantic functions* from the latter domain to the former, the latter domain is preferred as the domain of the representations, mappings between which are established by operations. *E.g.*, we would semanticize DDL programs as transformations on schemata (*syntax*), and not as transformations on such semantic representations as data base behavior. Thus, the domain of *operations semantics functions* is the following:

$$<\textit{Inf}^k \textit{ Operations Semantics Function}> = \\ [<\textit{Operations on } \textit{Inf}^k> \rightarrow \\ [<\textit{Synt } \textit{Inf}^k> \rightarrow \\ <\textit{Synt } \textit{Inf}^k> \times <\textit{Output}^k>]]$$

6. Principal Semantic Domains

The idea of *data semantics functions* has been presented in Section 4. This idea can be used to establish relationships between any pair of domains, one of which is *more semantic* for a particular user's purposes. The goal of this section is to elaborate such semantic domains that they would suit the purposes of most users of this approach to data base semantics.

Assume that for every relevant information level k the following conventional domains are given: a less-semantic domain of $\textit{Inf}^k$ informations' representations, $<\textit{SYNT } \textit{Inf}^k>$, a less-semantic domain of data base operations (activities) on this level ($<\textit{Operations on } \textit{Inf}^k>$), and a domain of all their possible outputs, $<\textit{Output}^k>$.

Example 6.1.

$<SYNT\ Inf^1>$ is the domain of all possible relational instantaneous data bases, i.e. collections of tables.

$<SYNT\ Inf^2>$ is the domain of all possible combinations of relational schemata, integrity laws, inference laws, exit routines, etc.

$<SYNT\ Inf^3>$ is the domain of all possible relational data base management systems.

$<Operations\ on\ Inf^1>$ is a union of data manipulation languages, query languages, etc.

$<Operations\ on\ Inf^2>$ is a union of data definition languages.

$<Output^k>$ for every k is a domain of all possible texts and dot-matrices which can be displayed on a terminal.

6.1. Semantic domains for non-first levels

Postponing the definition of a principal $<SEM\ Inf^1>$, the principal semantic domains for all other levels are defined in this subsection.

The definition is recursive. Assume that $<SEM\ Inf^k>$ has been defined.

Let inf_1^{k+1} be an information of Inf^{k+1}-th level. A semantic representation for inf_1^{k+1} must be explicit, comprehensive, simple, and its perception should not necessitate knowledge of information from higher levels of information.

inf_1^{k+1} induces a *data semantics function* on Inf^k level and *semantics-of operations function* on Inf^k level. Thus, these two things, and nothing else, should be extractable from the most semantic representation of inf_1^{k+1}. Thus we define a principal semantic representation of inf_1^{k+1} as a pair: a data semantics function on Inf^k and an operations-semantics function.

Thus:

$$<SEM\ Inf^{k+1}> =$$

$$(<Inf^k\ Data\ Semantics> \times <Inf^k\ Operations\ Semantics>) =$$

$$[<SYNT\ Inf^k> \rightarrow <SEM\ Inf^k>] \times$$
$$[<Operations\ on\ Inf^k> \rightarrow$$
$$[<SYNT\ Inf^k> \rightarrow <SYNT\ Inf^k> \times <Output^k>]]$$

Example 6.1.1.

$$<SEM\ Inf^3> =$$

$$[<SYNT\ Inf^2> \rightarrow ([<SYNT\ Inf^1> \rightarrow <SEM\ Inf^1>] \times$$
$$[<Operations\ on\ Inf^1> \rightarrow$$
$$[<SYNT\ Inf^1> \rightarrow (<SYNT\ Inf^1> \times$$
$$<Output^1>)]])] \times$$
$$\times[<Operations\ on\ Inf^2> \rightarrow$$
$$[<SYNT\ Inf^2> \rightarrow <SYNT\ Inf^2> \times <Output^2>]$$

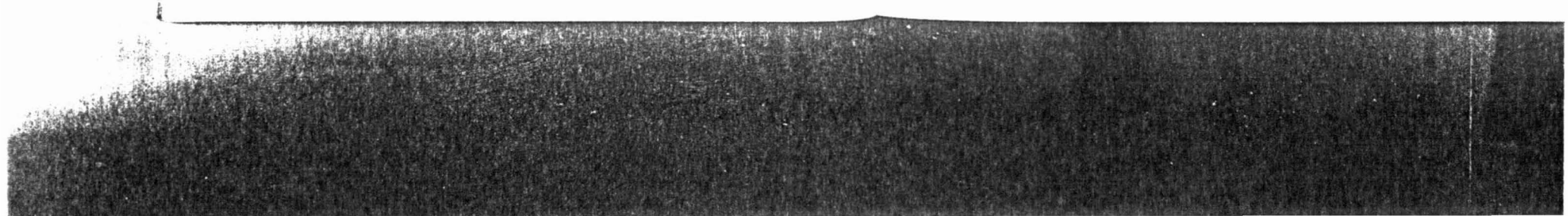
$\overset{e.g.}{=}$ for *example*

$$= [DATA-DESCRIPTIONS \rightarrow$$
$$([INSTANTANEOUS-DATA-BASES \rightarrow$$
$$INFERRED-INSTANTANEOUS-DATA-BASES] \times$$
$$[DATA-MANIPULATION-LANGUAGE \cup QUERY-LANGUAGE \rightarrow$$
$$[INSTANTANEOUS-DATA-BASES \rightarrow$$
$$INSTANTANEOUS-DATA-BASES \times$$
$$OUTPUT-TEXTS]])] \times$$
$$\times[DATA-DEFINITION-LANGUAGE \rightarrow$$
$$[DATA-DESCRIPTIONS \rightarrow$$
$$DATA-DESCRIPTIONS \times$$
$$OUTPUT-TEXTS]]$$

Remark.

The principal domain $<SEM\ Inf^{k+1}>$ is defined above as a Cartesian product. The representations belonging to this domain are only 97% comprehensive, i.e. bearing all their meaning in themselves. This is because in order to perceive them we use, *inter alia*, the following high-level knowledge:

- The first component of every pair is a data semantics function on Inf^k;
- for every less-semantic representation in $<SYNT-Inf^k>$, this data semantics function provides semantics as a structure belonging to the principal semantic domain for Inf^k, $<SEM\ Inf^k>$;
- the representations mapped to $\perp$ or *ERROR* are illegal; those mapped to *ERROR* can be detected by the software systems; those mapped to $\perp$ will cause the software system to loop infinitely when trying to detect the invalidity.
- The second component of every pair is a semantics-of-operations function on Inf^k;
- this function assigns every less-semantic representation of an operation on Inf^k with the semantics of that operation, i.e. a function which for every less-semantic representation of Inf^k gives a pair:
  - the first component thereof is the new less-semantic representation on Inf^k level produced by performing the operation on the former representation;
  - the second component thereof is an external output, printed or displayed;
- external inputs of operations, *e.g.* query parameters and standard input files of DML programs, are considered syntactic parts of operations themselves.



In order to increase the comprehensiveness of $<SEM\ Inf^{k+1}>$ by covering more of the foregoing high-level information, the definition of $<SEM\ Inf^{k+1}>$ can be refined using *abstract syntax notation* :

$<SEM\ Inf^{k+1}>$: :

$Inf^k-Data-Semantics$:
  ([$<SYNT\ Inf^k> \rightarrow <SEM\ Inf^k>$]),
$Inf^k-Semantics-of-Operations$:
  ([$<Operations\ on\ Inf^k> \rightarrow$
   [$Old-representation$:$<SYNT\ Inf^k> \rightarrow$
      $Operation-Outcome$:
        ($New-representation$: $<SYNT\ Inf^k>$,
         $External-output$: $<Output^k>$)]])

6.2. Semantic domains for level Inf^1

The choice of $<SEM\ Inf^1>$ depends on the requirements of the problem. Usually, a $SEM\ Inf^1$ would be a more user-oriented representation of an instantaneous data base than a $SYNT\ Inf^1$. Here are some possibilities:

1) According to ANSI-SPARC proposal of data base systems architecture three types of Inf^1 information representations should be considered: *internal* (organized according to *internal schemata*, convenient to the implementation and logically supportable by a DBMS), *conceptual* (organized according to a *conceptual schema* and being a comprehensive logical description independent of user-views and of the implementation aspects), and *external* (organized according to *external schemata* and convenient to users). It is very important to define and treat formally mappings between these representations. For this purpose, one representation (*e.g. internal*) can be defined as a $SYNT\ Inf^1$ and another (*e.g. conceptual*) can be defined as a $SEM\ Inf^1$.

2) When, for any other reason, one logical structure of data is represented by another, *e.g. binary* by *relational*, the latter can be considered a $SYNT\ Inf^1$, and the former a $SEM\ Inf^1$. *Inter alia*, this can be when a DML of one model is used with a data base of another model via a translator, or when the design of a data base was in the Binary model, but due to environmental reasons it has been implemented in a more data-dependent model.

3) $<SEM\ Inf^1>$ can be equal to $<SYNT\ Inf^1>$ and every considered Inf^1 semantic function can be a *semi-identity* function, i.e. mapping every $SYNT\ Inf^1$ to itself or to $\perp$ or to *ERROR*. This is when the only considered aspect of the Inf^2 is what Inf^1-s (or their representations) are permitted by data integrity laws.

4) $<SEM\ Inf^1>$ can be formally equal to $<SYNT\ Inf^1>$, yet some *data semantics* functions which are not *semi-identity* functions can be considered. *Inter alia*, they can represent

inference transformations with respect to knowledge in Inf^2 of *inference laws*.

Example 6.2.1.

Let $<SEM\ Inf^1> = <SYNT\ Inf^1> = X \times X \cup \{\perp\ ERROR\}$. Let inf^2 be one information of Inf^2 level. Let inf_1^2 imply that every information of Inf^1 is a transitive relation. Thus, inf_1^2 induces a data semantic function which maps every relation to its transitive closure. *E.g.*,

$$\{(x_1, x_2), (x_2, x_3)\} \in <SYNT\ Inf^1>$$

is mapped to

$$\{(x_1, x_2), (x_2, x_3), (x_1, x_3)\} \in <SEM\ Inf^1>$$

The latter representation is more comprehensive than the former. In order to perceive it less knowledge of inf_1^2 is needed.

Now let inf_2^2 be another information of Inf^2 level. Let inf_2^2 imply that every information of Inf^1 level is an irreflexive symmetric relation. Then the induced *data semantics function* maps $\{(x_1, x_2), (x_2, x_3)\}$ into $\{(x_1, x_2), (x_2, x_3), (x_2, x_1), (x_3, x_2)\}$, but $\{(x_1, x_1), (x_2, x_3)\}$ is mapped into *ERROR*.

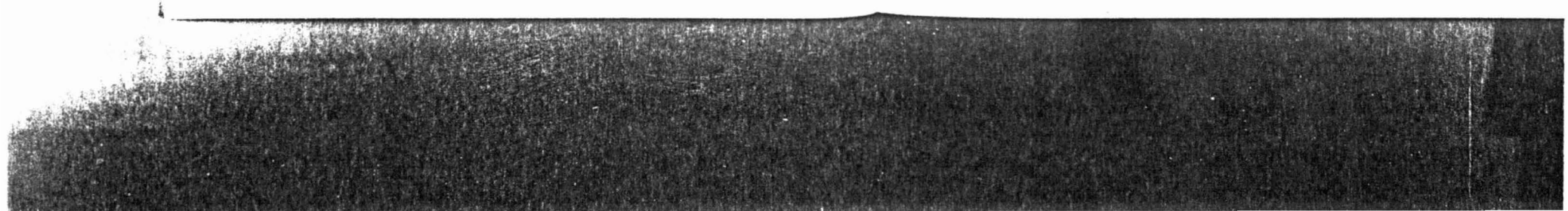
7. APPENDIX: Some Applications

Using the proposed approach, the following subjects have been studied in [Rishe-85]:

a) Data base integrity and inference laws expressible by assertions. They have been modeled, and their semantics has been defined as the behavior of data bases. Determinism and implementability of these laws have been investigated and several results have been proven demonstrating the power of the approach. Among the results is the following theorem: "Intersection-closed assertional inference laws are deterministic."

b) A version of the Conceptual Binary data base model has been developed and formally semanticized. This version has been based on data semantics, on data base integrity and inference laws and on their expressibility in data base schemata.

Categories and binary relations are treated in it uniformly according to their properties defined in schemata. These properties are treated as meta-relations and meta-categories, and are defined together with their meta-properties in a DBMS description, which is regarded as a meta-schema.



c) A data base model has been defined, having data manipulation and data definition non-procedural languages of maximal power: they can define every Turing-implementable query, update, integrity-law, inference-law.

This to covers the *relational*, *network* and *hierarchic* models which are shown in [Rishe-84] to be particular cases of the binary model.

7.1. Semantics of Assertional Inference and Integrity Laws in Data Bases

Data base inference laws are often expressed as assertions.

Example 7.1.1.

In a binary schema, the assertion "*STUDENT is a* **subcatregory** *of PERSON*" is an inference rule, a part of the inference law of the data base. If a fact "*x is a STUDENT*" has been stored in the data base, the system can infer: "*x is a PERSON*"

Unlike this, the assertion "*the categories PERSON* and *DEPARTMENT are disjoint*" is an integrity constraint, a part of an integrity law.

Such inference laws can be expressed syntactiely as programs yielding a Boolean value, or in a language equivalent to a predicate calculus, or in a schema of a data base.

Any of these representations can be converted into a slightly more semantic representation, which is a Boolean function over the domain *IDB* of all possible instantaneous data bases.

Yet, a more semantic representation of such an inference law is a transformation on *IDB*, i.e. a function: $[IDB \rightarrow IDB \cup \{ERROR, \perp\}]$. Taking into account also an integrity law, for every instantaneous data base this function should give a new inferred instantaneous data base, if possible.

The inferred instantaneous data base must satisfy the following conditions:

(1) It contains all the information represented in the original data base.

(2) It contains all the inferred information, i.e. satisfies the assertion of the inference law.

(3) It contains no extraneous information, i.e no proper subset of the inferred instantaneous data base satisfies (1)&(2).

(4) It satisfies the assertion of the integrity law.

The above definition is not deterministic. All the possible transformations corresponding to a given pair of assertional integrity and inference laws are formally defined in [Rishe-85]. Using an implementation-dependent enumeration of instantaneous data bases, one of the transformations is identified as a standard one. A function mapping every pair of laws into a standard transformation is constructively defined in [Rishe-85] using *lambda-calculus* and is

shown to be computable, i.e. implementable by a DBMS.

The determinism of inference laws is studied. An inference law is strictly-deterministic if for every two transformations corresponding to it, t_1 and t_2, and for every instantaneous data base idb, $t_1(idb) \in \{t_2(idb), \perp, ERROR\}$.

Theorem. An assertional inference law is strictly-deterministic if and only if it is *intersection-closed* (i.e. for every two instantaneous data bases satisfying the assertion of the law, their intersection satisfies the assertion too.)

7.2. Formalization of a simplified binary model defined by a metaschema

The following simplified binary model is formalized.

An instantaneous data base is a set of elementary facts. An elementary fact is either a binary fact: a relationship between two objects, or a unary fact: a statement that an object belongs to a category. The objects are either uninterpreted abstract objects or values.

The *less semantic representation* of an instantaneous data base is a set of basic elementary facts, i.e facts which have been reported by users via updates of the data base. The *more semantic* representation is an error-state (*ERROR* or $\perp$), or a set of all elementary facts: basic facts and facts which can be inferred from them.

The operations on instantaneous data bases are update transactions. A less-semantic representation of an update transaction is a pair of sets of elementary facts: a set of facts to be deleted from the data base and a set of facts to be inserted after the deletion.[7]

The least semantic representation considered of a data base description is a collection of meta-facts comprising the schema of a data base. These meta-facts define all the categories and the relations of the data base and their integrity properties (domains and ranges of relations, disjointness of categories, etc.) and inference properties (symmetry and transitivity of relations, the meta-relation *subcategory* between categories etc.)

This representation can be formally considered an instantaneous data base in itself and described by a *meta−schema*.

Example 7.2.1.

"*Jack is a student*" and "*Jack is a person*" are two facts.

[7] This representation is actually more semantic than a representation by a program. The reason to consider sets of facts rather than breaking a transaction into steps is the possibility of dissatisfaction of data base integrity when only a part of a transaction has been done.

"*STUDENT is a CATEGORY*" and "*STUDENT is a subcategory of PERSON*" are two meta-facts.

"*SUBCATEGORY is a transitive meta-relation between categories*" is a couple of meta-meta-facts.

The operations on data base descriptions are updates analogous to the update transactions of instantaneous data bases. Their semantics is deduced from the meta-schema.

The full formalization of the model is provided by finding the semantic object corresponding to the meta-schema. This semantic object is a principal semantic representation of Inf^3, composed of an Inf^2 data-semantics function and an Inf^2 semantics-of-operations function.

Its *data—semantics* component maps every schema to the behavior of a database.

The mapping is decomposed into three mappings as shown in

(i) A valid schema, which is a collection of consistent meta-facts, is mapped into a collection of all inferable meta-facts.

(ii) A collection of all inferable meta-facts is mapped into a pair of predicates over the domain semantic representations of instantaneous data bases: an integrity law, and an assertional inference law, specifying what semantic representations contain all the inferable facts.

(iii) The above pair is mapped into a principal semantic representation of Inf^2.

The four domains of Inf^2 representations considered in the above constitute a chain from a less-semantic domain to the principal semantic domain.

The semantics of the meta-schema is defined in [Rishe-85] constructively, using lambda-calculus, and it is proved there that the semantics is computable, and thus implementable by a DBMS.

7.3. Complete Semantics of a DBMS of an Absolutely Complete Power

[Rishe-85] proposed a non-procedural data-independent language model for data bases, in which *all* partial *Turing-computable* queries are specifiable. (As opposed to limited expressiveness of of "Codd-complete" languages, e.g. the relational calculus, and even of languages based on Horn-clauses, e.g. Prolog.) Every query is formulated in the language as a predicate calculus assertion expressing the desired relationship between the state of the data base, the information needed to be displayed, and auxiliary concepts. Interpretation of a query is a partial Turing-computable non-deterministic transformation which for any input state of the data base gives a minimal output to satisfy the assertion.

The language model proposed in [Rishe-85] yields query languages of complete computability power and also languages which are intended to restrict the use of undesirable or meaningless operations on objects. One of the important cases differentiates between abstract

objects, representing real-world entities, and concrete values. A more general case is parametrized by a family of permitted operations on the domain of objects and its subdomains. The languages are able to express every data base query reasonable within the restrictions parametrizing the languages.

The queries are syntaxed there as assertions, whose assertional semantic functions can produce *true*, *false*, or $\perp$ when applied to an instantaneous data base. The transformational semantics of queries, mapping instantaneous data bases to outputs or $\perp$ was defined there as follows.

$IDB \equiv INSTANTANEOUS\text{-}DATA\text{-}BASES$[8]

ASSERTION:

$[THE\text{-}LANGUAGE \rightarrow [IDB \rightarrow BOOLEAN_\perp]]$[9]

semantics-of-queries:

$[THE\text{-}LANGUAGE \rightarrow [IDB \rightarrow OUTPUT_\perp]]$

$$semantics\text{-}of\text{-}queries(\underset{\in THE\ LANGUAGE}{query}, \underset{\in INSTANTANEOUS\ DATA\ BASES}{idb}) =$$

let $\phi = ASSERTION(query)$ **in**

$$\text{choose}_\perp\left(\left\{(result, temp) \;\middle|\; \begin{array}{c} \phi(idb \cup result \cup temp) \wedge \overset{\forall vdb.}{\cup}{}^{10} \\ \textbf{if } idb \subseteq vdb \subset (idb \cup result \cup temp) \\ \textbf{then } \phi(vdb) \equiv false \end{array}\right\}\right)_{\text{project1 (take only } result \text{ component)}}$$

Semantics of queries was shown to be computable (equivalent to a partial recursive function), although this is not obvious from the above specification.

Here I shall use the language model proposed in [Rishe-85] to define a complete semantics of a data base management systems, covering ***all*** *Turing-computable* queries, integrity laws, inference laws, and update transactions (in

---

[8] For the relational model (having a denumerable domain D of objects),

$$IDB \equiv \underset{D}{\text{POWERSET}}(NAMES\text{-}OF\text{-}RELATIONS \times D^*)$$

Using the binary model, $D \cup D \times D$ is sufficient instead of D^*.

[9] $X_\perp$ is $X \cup \{\perp\}$; $\perp$ means ***undefined***.

[10] $\overset{\forall}{\cup}$ is a decomposable union: for $\alpha, \beta, \gamma, \delta \in IDB$, $\delta = (\alpha \overset{\forall}{\cup} \beta \overset{\forall}{\cup} \gamma)$ **iff** $(\alpha, \beta, \gamma) = \text{split}(\delta)$

the presence of integrity and inference laws).

semantics-of-operations:

$$[DATA\text{-}SEMANTIC\text{-}FUNCTIONS \rightarrow [IDB \rightarrow IDB_{\perp} \times OUTPUT_{\perp}]]$$

DATA-SEMANTIC-FUNCTIONS $=$

$$[IDB \rightarrow IDB_{\perp,ERROR}]$$

semantics-of-laws:

$$[THE\text{-}LANGUAGE \times THE\text{-}LANGUAGE \rightarrow DATA\text{-}SEMANTIC\text{-}FUNCTIONS]$$

$$\textbf{semantics-of-laws}((integrity, inference), idb) =$$
$$\in_{THE\ LANGUAGE\ THE\ LANGUAGE} \quad \in_{INSTANTANEOUS\ DATA\text{-}BASES}$$

let $\phi = ASSERTION(integrity)$, $\psi = ASSERTION(inference)$ **in**

let $inferred = idb \cup$

$$\text{choose}_{\perp}\left(\left\{(extension, temp) \;\middle|\; \begin{array}{c}\psi(idb \cup extension \cup temp) \wedge \\ \forall vdb. \\ \textbf{if } idb \subseteq vdb \subset (idb \cup extension \cup temp) \\ \textbf{then } \psi(vdb) \equiv false\end{array}\right\}\right)_{\text{project1}}$$

$$\textbf{in}(\textbf{if } \phi(inferred) \textbf{ then } inferred \textbf{ else } ERROR)$$

$$\textbf{semantics-of-operations}(datasem, operation, idb) =$$
$$\in_{DATA\text{-}SEMANTIC\text{-}FUNCTIONS\ (\text{consistent of integrity and inference laws})} \quad \in_{THE\ LANGUAGE} \quad \in_{INSTANTANEOUS\ DATA\ BASES}$$

let $(display, delete, insert) =$

$$\text{split}(query\text{-}semantics(operation, datasem(idb)))\ \textbf{in}$$

if $datasem((idb - delete) \cup insert) =_{\perp} ERROR$ [11]

then $(idb, ERROR)$

else $((idb - delete) \cup insert), display)$

The following definitions complete the full specification of a DBMS semantics.

The only type of data-definition operations in this language model is replacing and old pair of syntactic-specified integrity and inference laws with a new pair of laws.

[11] $=_{\perp}$ is equality which can be undefined: $(x =_{\perp} \perp) \equiv \perp$

Thus : $DATA\text{-}DEFINITION\text{-}OPERATIONS = THE\text{-}LANGUAGE \times THE\text{-}LANGUAGE$

$$\textbf{semantics-of-data-definition-operations}_{(new\text{-}integrity,\ new\text{-}inference)}(old\text{-}integrity,\ old\text{-}inference) =$$
$$((new\text{-}integrity, new\text{-}inference), \underset{(no\ output)}{\varnothing})$$

semantics-of-the-DBMS $\in$

$$[DATA-DESCRIPTIONS \rightarrow$$
$$([IDB \rightarrow INFERRED-INSTANTANEOUS-DATA-BASES] \times$$
$$[DATA-OPERATIONS \rightarrow [INSTANTANEOUS-DATA-BASES \rightarrow$$
$$INSTANTANEOUS-DATA-BASES \times OUTPUT]])] \times$$
$$\times [DATA-DEFINITION-OPERATIONS \rightarrow [DATA-DESCRIPTIONS \rightarrow$$
$$DATA-DESCRIPTIONS \times OUTPUT]]$$

semantics-of-the-DBMS $=$

$$(\lambda(integrity, inference) \in THE\text{-}LANGUAGE \times THE\text{-}LANGUAGE.$$
$$(\textbf{semantics-of-laws}_{(integrity, inference)}, \textbf{semantics-of-operations}_{(integrity, inference)}),$$
$$\textbf{semantics-of-data-definition-operations})$$

References

H. Biller, W. Glathaar, and E. Neuhold, "On the Semantics of Data Bases: The Semantics of Data Manipulation Languages," *Modelling in Data Base Management Systems, IFIP Working Conference on Modelling in DBMS's*, 1976.

M. Brodie and J. Schmidt, "Final Report of the Relational Database Task Group," *ANSI-X3-SPARC-DBTG, ACM SIGMOD RECORD*, vol. 12, no. 4, 1982.

J. M. Cadiou, "On Semantic Issues in the Relational Model of Data," *Mathematical Foundations of Computer Science, Proc. of 5th Symposium*, Springer-Verlag, 1976.

J. Clifford and D. S. Warren, "Formal Semantics for Time in Database System," *ACM Transaction on Database System*, vol. 18, 1983.

C. J. Date, "A Formal Definition of the Relational Model," *ACM SIGMOD RECORD*, vol. 13, no. 1, 1982.

Jay Earley, "On the Semantics of Data Structures," *Data Base Systems, COURANT Computer Science Symposium 6*, 1972.

L. D. Falkenberg, "Data Models: The Next Five Years," *INFOTECH Report: Data Base Technology*, vol. 1, 1978.

W. T. Hardgrave, "Ambiguity in processing Boolean Queries on TDMS Tree Structures: A Study for Four Different Philosophies," *IEEE Transactions on Software Engineering*, pp. 357-372, July 1980.

E. J. Neuhold and Th. Olnhoff, "Building Data Base Management Systems Through Formal Specification," *Springer LNCS #107*, 1981.

N. Rishe. [84] *Algorithms for Design of Relational, Network and Hierarchic Schemata from Conceptual Binary Schemata.* Tech. Rep. TRCS84-15, University of California, Santa Barbara, 1984.

N. Rishe. "Postconditional Semantics of Data Base Queries". Proceedings of the Conference on Mathematical Foundations of Programming Semantics (Manhattan, Kansas, April 1985).

N. Rishe. [85b] *Semantics of Universal Languages and Informations Structures in Data Bases.* Technical report TRCS85-010, Computer Science Department, University of California, Santa Barbara, 1985.

H. Schmid and S. Swenson, "On the Semantics of the Relational Data Model," *ACM-SIGMOD International Conf. on Management of Data*, San Jose, 1975.

J. Stoy, "The Scott-Strachey Approach to Programming Language Theory," *Denotational Semantics*, 1977.

Y. Vassiliou, "Null values in database management: A denotational semantics approach," *Proc. of: ACM-SIGMOD international conference on Management of data*, pp. 162-169, Boston, May 30 - June 1 , 1979.

H. Weber, "A Semantic Model of Integrity Constraints on a Relational Data Base," in *Modeling in Data Base Management Systems, IFIP Working Conference on Modeling in DBMS's*, ed. G. M. Nijssen, pp. 162-169, Boston, May 30 - June 1 , 1976.

C. Zaniolo, "Database Relations with Null Values," *Journal of Computer and System Sciences*, vol. 28, pp. 142-166, 1984.