ON FORMAL SEMANTICS OF DATA BASES

by

Naphtali Rishe

Technical Report #198

February 1981
ON FORMAL SEMANTICS OF DATA BASES.

by Naphtali RISHE.

ABSTRACT.

An approach to defining formal denotational semantics of data bases and db management systems is presented and its objectives are discussed. Semantics of instantaneous data bases, of db schemas, of DBMS, etc. are distinguished and generalized. Semantics of general laws of data integrity (systems of logical constraints in a db) and information inference is studied. A simplified version of a conceptual binary db model, whose behavior is determined by integrity and inference laws, is presented with its denotational semantics.

Keywords and phrases: Data base semantics, Data bases, Conceptual schema, Integrity, Inference, Representations of information, Binary db model, Denotational semantics, Vienna Development method (VDM).

This article presents a new approach to defining formal semantics of data bases (db) and db management systems. Data bases can be better understood and formally treated using it. The approach is applied to study of a data base model whose behavior is controlled by laws of data integrity and of information inference.

Chapter 2 deals generally with db semantics.

Common opinions on db semantics are surveyed in section 2.1. Some objectives of defining db semantics are discussed in section 2.2. Different levels of information related to a db system are discussed in section 2.3. They have different influences and meanings (semantics), but the handling of these levels is unifiable.

Different representations of information (related to a data base or a DBMS) are discussed in section 2.4. Semantic versus syntactic representations are distinguished there with respect to their relative comprehensiveness and comprehendability. Mappings between the representations controlled by some higher-leveled information are defined there as data semantics functions.

Being distinguished from data semantics, auxiliary semantics of operations is regarded in section 2.5. of operations is regarded in section 2.5. Domains for semantic representations of information of all levels are proposed in section 2.6.

Several notes on the Extended Abstract Syntax Notation, which is convenient for specification of Syntactic and Semantic domains and is used in this paper, are given in section 2.7.
Chapter 3 studies the semantics of a conceptual db model with laws of data integrity and of information inference.

A variant of the conceptual db model is introduced in section 3.1.8.

General classes of a conceptual model are described in section 3.1.9.

General integrity and inference laws and their semantics are studied in section 3.1.10. Some results on determinism and computability of db transformations, implied by the laws, are proved there.

An approach based with laws expressed in a schema and in a meta-schema is presented and semanticized in section 3.5.
2. AN APPROACH TO DB SEMANTICS

2.1 COMMON VIEWS ON DB SEMANTICS

The presented here approach to db semantics has been influenced by the ideas of denotational semantics of programming languages, by Bjorner's approach to semantics of db operations and by databases' regarding of user's-world-oriented db models as semantic models, and of logical db constraints and information implication rules as semantic aspects of data bases.

Here are some characteristic papers of those common views.

[Earley-72a] 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-75a], regard data base semantics as a system of constraints on information that can be stored in the db.

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

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

According to [Falkenberg-78a], "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."
Formal treatment of db semantics appears in [Weber-76a], who defined a formal [Syntactic(*)] model of constraints and assigned to it semantics in the Operational style; in [Biller-76a], who defined denotational semantics of db schemas, mapping them to sets of world states, and of a db manipulation language, mapping its programs to pairs of world states; and in [Bjorner-78b], who defined mappings from db operations and abstract states of the db to its new states and output.

In our approach we generalize and refine those views on db semantics. Generalizing, we distinguish between semantics of a data base and semantics of operations on it, and between semantics of an instantaneous db, db Schema/Application, DBMS, etc.

2.2 SOME OBJECTIVES OF DEFINING FORMAL DENOTATIONAL SEMANTIZATION OF DATA BASES.

Following is a list of some possible purposes of defining formal semantics for data bases and DBMS.

1) To provide meaningful formal description of

(*) - We shall describe later what we mean by the term 'Syntactic'.

---------
- DBMS and its DDL, DML and Query languages;
- the Schema and the administrating software complex (incl. exit routines, etc.) of a data base;
- the Subschema/Userview and the software complex of an application;
- instantaneous data bases;
- etc. for other levels of db information.

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

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

3) To achieve a degree of automatization in constructing (or programming, generating, etc.) of
- instantaneous data base (or a part thereof), when the meaning of the information which should be represented in it is given;
- db schema, subschema and (may be as a utopia) db administering and application software, according to a specification of the needed properties;
- (maybe as a utopia) DBMS, according to requirements from it;
- etc.
4) To provide means for verification of certain constructs (such as schemas, syntactic specifications of integrity laws, system parameters, etc.) against the needed properties of the behavior of a database.

5) To study desired properties of design of DDL, DML, and other related languages.

5) When desired properties of db behavior cannot be satisfied (implemented) by any construct (or software), to find the constructs (or software) which APPROXIMATELY (as must be defined) satisfy the properties. Among these, a best (with respect to certain criteria) can be chosen.

DBMS and/or programming in the db.

2.3. LEVELS OF INFORMATION IN A DB SYSTEM.

2) To study properties of constructs in order to treat data base semantics, we have to distinguish between three levels of information related to a data base system. The following is a preliminary discussion, not aiming yet to define formal semantics. Neither information representations nor their semantics are considered here.

It is customary to classify db information according to levels of its descriptivity. Usually, three levels are distinguished. (*)

These three levels are the following.

1) INSTANTANEOUS information, represented by an instantaneous database.

(*) E.g., in [DB1:80a].
2) I*2 -- the information about the common properties of I*1 information which can be represented at any instant of time by a given database (provided the properties and the purpose of the database are kept constant). A part of an I*2 information is represented by the schema of a database. Another part can be represented by an additional "semantic" schema and/or just be known to the systems and users. An I*2 information can contain/impose certain laws on a database, some of them are of the following types:

- Integrity Laws -- specifying principally what are the valid states of this database;
- Inference Laws -- specifying how from information entered to the database other information can be deduced (by the DBMS or elsewhere);
- Laws Of Operations -- specifying what operations (incl. atomic, complex, or whole processes, sessions, etc.) are permissible on the database and what are their results depending on the states of the db. A law of operations includes an Update law.

These laws and other parts of an I*2 information can be accumulated in Exit routines, basic application programs, etc.

3) I*3 -- the information describing all possible I*2 informations which a database can possess under a given DBMS. An I*3 information can be considered as accumulated (at least partly) in the code of a DBMS.

For some purposes, more than three levels may need to be considered. E.g. the above I*1 level can be splitted into two: the lower corresponding to less General/Stable data in the db, and
the higher corresponding to more General/Stable data; the above $I^*2$ level can be split into a lower level corresponding to subschemas and a higher level corresponding to schemas; the above $I^*3$ 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.).

Let us assume that, for a certain purpose of a user of our approach to formalization, n levels of information need to be considered.

For any $k$ from 1 to $n-1$, any $I^*k+1$ Information describes a whole range of $I^*k$ "Informations". At the highest level $n$ considered, only one specific Information is of interest.

2.4 REPRESENTATIONS OF INFORMATION AND DATA SEMANTICS.

<Operation on $I^*k$>, whether it is legal, and if so, how it would change the $k$-th level of the db and what external output it would produce depending on the present $I^*k$. 

A. Domains of representations.

Thus, from the $I^*k$ we can extract information.

Let us consider now representations of Information of some $I^*k$ function:

$\langle\text{Operation on } I^*k\rangle \rightarrow$ level ($0<k<n$). Any $I^*k$ Information can be represented in different ways: implicit, explicit, computer-oriented, end-user-oriented, db-specialist-oriented, denoted by known terms, mathematically abstracted, Denotatively Semantized, etc.

Comprehending and interpreting any representation usually necessitates knowledge of an $I^*k+1$ 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 the semantics of $I^*k$ alone, it would
view of his purposes and his exploitation and perception of them, one of them can be more comprehensive, clear and usable (subjectively for him), and its perception can be less dependent on a deep knowledge of the other than that of the other. In this case we would call the former representation More Semantic, and the latter More Syntactic. Our distinction between Syntactic and Semantic representation is comparative and not absolute. A chain of representations, from a Very Syntactic to a Very Semantic can be considered.

Examples,

1. A certain I*2 can be represented by a schema. This representation is Syntactic 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 db. (One can learn on this behavior from a representation of the I*2 and the known I*3.)

2. The I*3 information accumulated in the software of a certain DBMS, can be represented by the string 'System-R Vers.1 Rel.1', which is perceivable with the use of the I*4, which, inter alia, relates all identifications of DBMS-es to their full descriptions.

Another, More Semantic, representation of the same I*3 is a user following is a list of some possible purposes of defining formal manual for this DBMS (for the perception of which the user "less semantics in data bases and DBMS

(*)- In our nomenclature, "syntax" is just an opposite to "semantics", and no strictness or string or language properties are necessarily associated with a More Syntactic representation. Yet, the origin for these terms is in the theories of programming language semantics, where--the non-semantic representation -- a program (whose perception depends on understanding of the language), i.e., Higher I* is syntactic.
needs to consult the I*4). Actually, the latter representation can be inadequate for many purposes of semantization. A better representation is discussed in section 2.6.

(End of examples.)

When a set (*X) of I*4 informations is of interest, for every one of which there is a More Semantic representation and a more

Syntactic representation, two domains(X) can be considered, one containing (inter alia) the More Syntactic representations, and the

other, the More Semantic representations. Let us denote these domains <Synt I*4> and <Sem I*4> respectively. (a).

B. Data semantics functions.

Let ik10 be a given I*4k information, and let there be two given
domains of representations of I*4k informations, one of them chosen to contain the 'syntactic' representations and the 'other' -- the

semantic' ones, called <Synt I*4k> and <Sem I*4k> respectively.

From ik10 one can learn, inter alia, the following knowledge:

a) what Synt I*4k-s represent I*4k permissible according to the

ik10;

b) what Sem I*4k-s represent I*4k information permissible

(Alternatively: implementable by a Synt I*4k or representable by a

Sem I*4k).

(*X) - 'not a set' in the mathematical sense, but an intuitively

quasi-set of Non-representations.

(%) - 'for the meanwhile, a domain is just a mathematical set.'

(a) - In this paper we use the following convention for names:

'Nnnn' (or 'NNNNN') is a concept (whose meaning in this paper may
differ from others' definitions), e.g. 'Integer'. 'Nnnn' can be

used in a sentence to categorize an entity: 'an Nnnn', 'the Nnnn'.

<Nnnn> (or <NNNNN>) is the set (or the domain) of the Nnnn-s, e.g

<integer> is the set of all integers, 

[0, 1, 2, 3, ...] are fixed

members of this set, e.g. in the phrase 'let integer be the least

positive integer'.


Synt \( I^*k \) according to the ik10;

3) what is the correspondence between the valid elements of \(<\text{Synt } I^*k>\) and the valid elements of \(<\text{Sem } I^*k>\).

This knowledge is expressed in a Data Semantics Function (*) which maps every \(<\text{Synt } I^*k>\) into its data semantics, i.e. the corresponding \(<\text{Sem } I^*k>\), if there is any. This function can be just a partial function from \(<\text{Synt } I^*k>\) to \(<\text{Sem } I^*k>\).

We shall revise this definition and the definition of domains in order to

- 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 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, DPL, Eel procedures, etc.

We now assume that any considered domain is a set with a complete partial order \( \leq \), relating sets, defined to better defined elements, having the minimum element called \( \bot \) and having a special element called \( \top \) or meaningless or ERROR. Any simple

(\( \top \)). The term Data Semantics is used in order to distinguish this from the latter discussed semantics of operations \( \ast \) as defined above in Example 1.

Technion - Computer Science Department - Technical Report CS0198 - 1981
set (or a simple domain without CPO) that needs to be considered is convertible to a domain satisfying the above by adding to it two special elements, one with the flat order \((*)\). Any simple set (domain) which will not be assumed to have a CPO will be named ending with \(\text{P.Proper} \). 

We now define the domain containing all the possible \(\text{Data Semantics Functions} \) (corresponding to different \(I^{*k+1}\)) from \(\text{Synt } I^{*k} \) to \(\text{Sem } I^{*k} \) as the domain of all the Continuous (def. of continuity see, e.g., in [Stoy-77a]) functions between these domains. 

\textbf{Data Semantics Function (from } \text{Synt } I^{*k} \text{ to } \text{Sem } I^{*k} \text{ ) in programming languages: } \text{Synt } I^{*k} \rightarrow \text{Sem } I^{*k} \text{ (evaluation) \((*)\)}

A syntactic function \(F \) is mapped by a function into \(\text{Sem } I^{*k} \) or \(\text{Synt } I^{*k} \) is assumed to be illegal according to the corresponding \(I^{*k+1} \). The difference is in the ability of the system to detect the invalidity. Usually \(F \) would denote a detectable invalidity, while \(I \) -- the one which would cause the detecting software to loop infinitely. 

\(\text{2.5 Auxiliary Semantics of Operations.} \)

We need not to be the supremum, unlike the notation in some other sources (e.g., [Stoy-77a]).

\((*)\) - The flat order over the domain \(D \) with minimum \(1 \) is \(\{(x, y) \in D^{**2} \mid x \equiv 1 \vee x \equiv y \}\). 

\((\star)\) - This domain of functions has the pointwise \(\sup \) meaning that one semantics function \(\text{less} \) \(\text{defined} \) than another or \(\text{approaches} \) it as defined, e.g., in [Stoy-77a].
In our approach, the Semantics Of Operations is auxiliary (relatively to Data Semantics), yet it needs to be discussed. Let <Operation on I*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 db level, the domain can be equal to the DML, and for the db/Schema level it can be equal to the DDL.

We regard external input to an operation as an integral part of it. Let <Output-k> be a domain of external output from the operations of <Operation on I*k>.

For a given I*k+1 ik10, it is known for every operation <Operation on I*k>, whether it is legal, and if so, how it would change the k-th level of the db and what external output it would produce depending on the previous status of the db.

Thus, from the ik10 we can extract an Operations Semantics Function:

\[
\text{Function :} \\
\text{[<Operation on I*k> \rightarrow} \\
\text{[<Representation of I*k> \rightarrow} \\
\text{<Representation of I*k> x <Output-k>]]}
\]

We have to choose what <Representation of I*k> is suitable for the above transformation. As we separated the semantization of data to Data Semantics, it is desirable to consider only Syntactic db transformations in the semantics of operations. E.g., we would
semanticize DDL programs as transformations on schemas (Syntax) as it is comprehended by the db users, and not as transformations on such semantic representations as db behavior.

Thus, the domain of Operations Semantics Functions is following:

\[
\text{<Operation on } I^*k> = \text{<Synt } I^*k> \times \text{<Output-}k>.
\]

Our view of Semantics Of Operations much resembles Bjorner's db semantics (*), though in our approach it is only auxiliary semantics of operations, and the principal semantization is in Data Semantics.

2.6 DESIRABLE SEMANTIC DOMAINS.

We propose now desirable domains for semantic representations of db information. By 'desirable semantic domain' is intended a domain of representations which would suit the aforementioned purposes of semantization when the choice of the domains is not dictated by semanticizer's specific purposes.

Assume, that for every information level of interest k the Syntactic domains <Synt I^*k>, <Operation on I^*k> and <Output-}k> are given.

(*)- [Bjorner-79a] defines db semantics as a function from operations and old states of the db \((I^*1+I^*2\) in the notation used by us) to the new states of the db and output.
A. We postpone the problem of choice of \(<\text{Sem } I^*k+1>\) and define \(<\text{Sem } I^*k+1>\) recursively assuming that \(<\text{Sem } I^*k>\) has been defined.

Let \(k\) be a positive integer, \((<k+1>)\). Let \(k_0\) be an \(I^*k+1\). We seek a semantic representation for \(I^*k_0\), which is explicit, comprehensive, simple, and whose perception does not necessitate knowledge of information from higher levels. As we stated before, there are two things extractable from \(I^*k_0\), and we want them to be directly extractable from its comprehensive semantic representation. Thus, we compose this representation of the corresponding (to \(I^*k_0\)) \(I^*k\) Data Semantics Function and Semantics Of \(I^*k\) Operations. The domain containing such representations for all possible \(I^*k+1\)-s (provided the Syntactic domains are fixed), follows:

\(<\text{Sem } I^*k+1> \triangleq <\text{Data Semantics Function (from }<\text{Synt } I^*k> \text{ to }<\text{Sem } I^*k>> \times <\text{I^*k Operations Semantics Function}>\)

B. The choice of \(<\text{SEM } I^*1>\) depends on the requirements of the problem. Usually, a \(\text{SEM } I^*1\) would be a more user-oriented representation of information than a \(\text{SYNT } I^*1\). Following are some examples.

1) According to ANSI-SPARC proposal of db system architecture (which is discussed e.g. in [Braccchi-76b], [Falkenberg-78a], [Infotech-78a], [Steel-78a], [Adiba-78a]), three types of \([I^*1]\)

\((\triangle)\) - the symbol '\(\triangle\)' means 'is defined to be'.

\((\*)\) - We feel that our intention will be better expressed in Extended Abstract Syntax Notation which will be introduced later. The better definition is:

\(<\text{Sem } I^*k+1>: : <\text{Data Semantics Function (from }<\text{Synt } I^*k> \text{ to }<\text{Sem } I^*k>> \times <\text{I^*k Operations Semantics Function}>\)

\((\#)\) - The domains on the right side have been defined in sections 2.4 and 2.5 (depending on \(<\text{Sem } I^*k>\) and Syntactic domains).
information representations should be considered in a db system:
Internal (organized according to Internal Schemas, 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 Schemas 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 I*1 and another (e.g.
Conceptual) can be defined as a SEM I*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 as a SYNT I*1, and the former -- as a SEM I*1.

3) <SEM I*1> can be equal to <SYNT I*1> and every considered I*1
semantic function can be a Semi-identity function, i.e. mapping
every SYNT I*1 to itself or to υ or to ERROR. This is when the
only considered I*1 Data Semantics aspect of the I*2 is what I*1s
for their representations) are permitted (due to Data Integrity
examples.

4) <SEM I*1> can be formally equal to <SYNT I*1>, yet some data
semantics which are not Semi-identity functions can be considered:
E.g. let <SEM I*1> = <SYNT I*1> Adib (1980) Let the ~I*2 I20
state that I*1 is an Equivalence Relation. Then the SYNT I*1

I*10 = (x1, x2, x3) can be mapped by the Data Semantics
Function into the SEM I*1 = {(x1, x1), (x2, x2), (x3, x3),
(x1, x2), (x1, x3), (x2, x3)}, which is more comprehensive than sy10

and whose perception less necessitates knowledge of the I*2. If
(#)- The domains on the right side have been defined in a way
and I27 states that I*1 is an irreflexive relation, then the
2.7 EXTENDED ABSTRACT SYNTAX NOTATION

When semantic representations are investigated or syntactic representations are semanticized, it is usually irrelevant how these representations are syntactically-sugared. A very convenient means for defining a domain, neglecting syntactic-sugaring aspects of its elements, is the Abstract Syntax Notation, which is defined e.g. in [Bjorner-78a]. However, as we deal with ordered domains, we need to extend those definitions for C,P ordered domains. We shall not fully introduce the extension here, but shall only discuss those aspects thereof which are used in the definitions in the following sections.

When an equation of the type

\[ <R_1> :: <R_1> <R_2> <R_3> \ldots <R_k> \]  

appears in a system of Abstract Syntax equations defining a group of domains, it means in Abstract Syntax Notation that there exists a bijection \( \text{mk-<R>} : (\langle R_1 \rangle \times \langle R_2 \rangle \times \ldots \times \langle R_k \rangle) \text{bij} \rightarrow \langle R \rangle \) \((\dagger)\). The projections of its inverse are denoted \( \text{select-<R_i><R>} : <R> \rightarrow \langle R_i \rangle \).

\((\dagger)\)-the names of the domains on the right side must be distinct, \(i.e.\) \(i,j\) if \(0<i<j<k+1\) then \(\langle R_i \rangle \sim \langle R_j \rangle\), but the domains themselves need not be distinct.

\((\dagger)\)-Its application can be written as

\[
\text{mk-<R>} (\text{select-<R_1>} : r_1, \\
\text{select-<R_2>} : r_2, \\
\ldots, \\
\text{select-<R_k>} : r_k)
\]
Constructive implementations exist such that the occurrence of the domains on the right side is associative, e.g. the following domains are equal:

\[
\langle A \rangle :: \langle B \rangle \langle C \rangle
\]

\[
\langle E \rangle :: \langle C \rangle \langle B \rangle.
\]

In the Extended Abstract Syntax Notation we require also that if the domains on the right side have CPO's \(1, 2, \ldots, k\) then the left-side domain has CPO satisfying:

\[
m_k-<R>(\ldots, \text{select}<R_i>: r_{i1}, \ldots \text{select}<R_k>: r_{k1}) [\]
\[
m_k-<R>(\ldots, \text{select}^2<R_i>: r_{i2}, \ldots \text{select}<R_k>: r_{k2}) \iff
\]
\[
L_i \leq 4_i (0 \leq i \leq k+1), r_{i1} [i \ r_{i2}
\]

Another aspect of Extended Abstract Syntax Notation used in this paper is the possibility of having, among the domain-defining equations, equations of the type:

\[
<R> = [<R_1> \to <R_2>],
\]

meaning, that \(<R>\) is the domain of all continuous functions from \(<R_1>\) to \(<R_2>\):

Some aspects of Abstract Syntax Notation will be explained if and when they are used in the following sections.

It is quite convenient to mathematically operate with representations (from domains defined by Abstract Syntax Notation), using abbreviations for mathematical functionals and formulae that are defined in VDM Meta Language ([Bjorner-78a]). When these abbreviations are not obvious, they will be commented upon. Most of the operators (functionals) will be assumed to be extended to Normally propagate \(1\) and \(\text{ERROR} (\emptyset)\).
Constructive implementations exist such that the occurrence of the domains on the right side is associative, e.g. the following domains are equal:

\[ <A> :: <B> <C> \]
\[ <B> :: <C> <B> . \]

In the Extended Abstract Syntax Notation we require also that if the domains on the right side have CPO's \([1, 2, \ldots, k]\) then the left-side domain has CPO \([\_\text{satisfying:} \]
\[ \text{mk-}<R>(\ldots, \text{select-}<R_i>: r_{i1}, \ldots \text{select-}<R_k>: r_{k1}) \]
\[ \text{mk-}<R>(\ldots, \text{select-}<R_i>: r_{i2}, \ldots \text{select-}<R_k>: r_{k2}) \text{ iff} \]
\[ \forall i (0 \leq i < k+1). r_{i1} [i \mid i2 \]

Another aspect of Extended Abstract Syntax Notation used in this paper is the possibility of having, among the domain-defining equations, equations of the type:

\[ <R> = [<R_1> + <R_2>], \]

meaning, that \(<R>\) is the domain of all Continuous functions from \(<R_1>\) to \(<R_2>\).

Some aspects of Abstract Syntax Notation will be explained if and when they are used in the following sections.

It is quite convenient to mathematically operate with representations (from domains defined by Abstract Syntax Notation), using abbreviations for mathematical functionals and formulae that are defined in VDM Meta Language ([Bjorner-78a]). When these abbreviations are not obvious, they will be commented upon. Most of the operators (functionals) will be assumed to be extended to normally propagate \(\downarrow\) and ERROR (\(\varepsilon\)).
Constructive implementations exist such that the occurrence of the domains on the right side is associative, e.g. the following domains are equal:

\[<A> :: <B> <C> \]
\[<E> :: <C> <B>.\]

In the Extended Abstract Syntax Notation we require also that if the domains on the right side have \(\text{CPO's }[1, [2, ... , [k\) then the left-side domain has \(\text{CPO}\) satisfying:

\[
\text{mk-} <R>(..., \text{select-}<R_i>: r_{i1}, ..., \text{select-}<R_k>: r_{k1}) [\text{mk-} <R>(..., \text{select-}<R_i>: r_{i2}, ..., \text{select-}<R_k>: r_{k2}) \iff \]
\[
i_j (0 < i < k+1). r_{i1} [i r_{i2} .
\]

Another aspect of Extended Abstract Syntax Notation used in this paper is the possibility of having, among the domain-defining equations, equations of the type:

\[<R> = [<R_1> \rightarrow <R_2>],\]

meaning that \(<R>\) is the domain of all continuous functions from \(<R_1>\) to \(<R_2>\).

Some aspects of Abstract Syntax Notation will be explained if and when they are used in the following sections.

It is quite convenient to mathematically operate with representations (from domains defined by Abstract Syntax Notation), using abbreviations for mathematical functionals and formulae that are defined in \(\text{VDM Meta Language}\) ([Björner-78a]). When these abbreviations are not obvious, they will be commented upon. Most of the operators (functionals) will be assumed to be extended to Normally propagate \(1\) and \(\text{ERROR (\$)}\).
3. SEMANTICS OF A CONCEPTUAL BINARY DB MODEL

3.1. A VARIATION OF A CONCEPTUAL BINARY DB MODEL

A. Sources and motivation.

The Conceptual Binary Model (*) is the model of information representations which can be viewed as collections of primitive facts of two types: stating that there is a relation between two entities, or stating that an entity is of certain category. This basic model has been much investigated lately in three directions.

1) According to ANSI-SPARC PROPOSAL (2), the Universe of Discourse (an enterprise information) should be mapped by the Enterprise Administrator to a formal Conceptual description, which is a comprehensive logical data representation independent both of implementation aspects and of views of particular users (in a multi-user system), convenient and "natural" to the Enterprise Administrator, and highly flexible. It was argued by many authors (e.g. [Bracchi-76b], [Kent-76a]), that the most suitable model for Conceptual Schema and Description is the Conceptual Binary model (in the aforementioned meaning of this term).

2) A Conceptual Binary Model was extensively used (e.g. [Schmid-75a]) for an Auxiliary Semantic Schema specifying constraints in a Relational data base, though there is a tendency

-----------
(2) - Except for 'if then else' e.g., which propagates not Normally.
(*) - This term may be understood differently by some authors.
(2) - It was mentioned in section 2.6.
(e.g. [Flory-78a]) to prefer for this purpose Chen's Entity-Relationship model ([Chen-76a]), which is much less Data-independent and much more Implementation-oriented.

3) It was argued that the Conceptual Binary model is very desirable as a basic db model (e.g. [Abrial-74a], [Vandijk-77a]) and is much more Data-independent, natural and convenient than the other models such as Hierarchic (e.g. [DB1-80a]), Network (e.g. [Codasyl-71a]), Relational ([Codd-70a]) and even Binary models having additional structure elements (e.g. distinguishing between a Relation and an Attribute), such as Sēnko's model ([Senko-75a]).

Any other model's representation of $I^*1$ and $I^*2$ information can be easily and naturally converted to a representation according to the Conceptual Binary model. The contrary is not the case generally.

Examples (based on discussions in [Abrial-74a] and [Vandijk-78a]):

The $I^*2$ represented in the Relational model as the Ternary Relation MET over the domains MAN, MAN, TIME, can be represented in the Conceptual Binary model as the Categories MEETING, MAN, TIME, the m:2 Binary Relation MEETER From MEETING To MAN and the Functional Binary Relation TIME-OF-MEETING From MEETING To TIME; the $I^*2$ represented in the Relational model as the relation ORDER over the domains ARTICLE, MAN, QUANTITY can be represented in the Conceptual Binary model as the Categories ORDER, ARTICLE, MAN, QUANTITY and the Functional Binary Relations ORDERED-ARTICLE, ORDERING-CLIENT, ORDERED-QUANTITY. Also, these relations will usually be between abstract categories and not between domains of values (as it is in the Relational Model), so there can be
additional Value Relations, e.g SOCIAL-SECURITY-> From MAN. To
INTEGER.

B. The variant investigated here.

An intuitive sketch of the investigated here variant of the
Conceptual Binary Model follows. A formal description will be
given in the next section.

Any $I^*1$ is represented by a finite collection of Facts (as stated
above). Neither Relations nor Categories need be Disjoint. Some
Entities are Abstract and some are Values. Some of the Relations
relate entities to "values", thus simulating "attributes".

A part of any $I^*2$ information is represented in a schema
specifying the permitted Relations, Categories and their
interrelations. This covers a part (Structural) of the integrity
constraints and a part of Information inference rules. Other
parts need to be expressed by other $I^*2$ constructs, which we shall
not discuss here (yet certain general means will be discussed in
section 3.4). A Syntaxically-Sugared example of a schema
follows:

```
Trader is a Category(check!);
Organisation is a Category(check!);
Company is a Subcategory(infer!) of Trader and of Organisation;
Sale is a Category(check!);
Buyer is a Function From Sale To Trader;
Name is a Relation From Trader To String;
```

A schema represents Meta-information which can be updated like any
3.2 SYNTACTIC AND SEMANTIC REPRESENTATIONS OF INFORMATION OF DIFFERENT LEVELS.

A. Representations of $I^*$.

1) SYNT $I^*$

A basic syntactic representation of information will be a system of elementary facts (*). These are those facts which have been entered to the data base by users, not rejected by the system and not nullified later. Actually, this representation is quite abstract and is rather semantic from the point of view of data-independence and self-expressability. Yet, in our

(*) or more precisely: of Representations Of Elementary Facts, which will not be distinguished from the Elementary Facts themselves (for simplicity).
nomenclature, we call it syntactic relatively to later defined semantic representations.

Thus, postponing the definition of the domain of Systems of Facts, we define:

\[ <\text{SYNT } I^*1> = <\text{System } \text{of } \text{Facts}> \]

2) \%EM \%1

By any given \%1 Data Semantics function, any given \%YNT I*1 representation is mapped to a Semantic representation, which is either

\ a) \% or ERROR denoting invalid data base,

\ or, for a valid data base:

\ b) a system of facts which can be deduced from the \%YNT I*1.

The \%EM I*1 does not have to be stored in the computer system (even the \%YNT I*1 does not).

For a given \%YNT I*1, the \%EM I*1 is supposed to be known to the software system and to users. It can either be or can not be directly retrievable from the data base. This ability depends on the semantics of the db retrieval operations, which will not be discussed in this paper.

We can define the domain of \%EM I*1 as follows:

\[ <\text{SEM } I^*1> = <\text{System } \text{of } \text{Facts}> \]

B. Representations of \%2

1) \%EM \%2:

The \%EM I*2 is standard, i.e.

\[ <\text{SEM } I^*2> : \%I^*2 \\text{Data } \text{Semantics } \text{function} > <\text{Semantics of} \]

\text{...}
Operations over \( I^*1 \):

\[
\langle I^*1 \text{ Data Semantics Function} \rangle = [\langle \text{SYNT} \ I^*1 \rangle \rightarrow \langle \text{SEM} \ I^*1 \rangle]
\]

As for the operations, we shall restrict our discussion to transactions only. Thus,

\[
\langle \text{Semantics Of Operations over} \ I^*1 \rangle =
\langle \text{Semantics Of Transactions over} \ I^*1 \rangle =
\]

\[
[\langle \text{Transaction} \rangle \rightarrow [\langle \text{SYNT} \ I^*1 \rangle \rightarrow \langle \text{SYNT} \ I^*1 \rangle \rightarrow \langle \text{Output} \rangle]]
\]

Specification of a \( \text{SEM} \ I^*2 \) fixes a correspondence between the domains \( \langle \text{SYNT} \ I^*1 \rangle \) and \( \langle \text{SEM} \ I^*1 \rangle \).

2) Abstract Integrity and Inference laws.

In section 3.4 we shall discuss and semanticize rather general and abstract representations of \( I^*2 \): Data Integrity and Information Inference Laws.

These representations are "syntactic" relatively to the more explicit and comprehensive \( \text{SEM} \ I^*2 \) representations, but they are semantic relatively to some other representations that are discussed later.

These representations are called \( \text{SYNT-SEM-SEM} \ I^*2 \), and their domain is denoted \( \langle \text{SYNT-SEM-SEM} \ I^*2 \rangle \). In section 3.4 we shall define this domain precisely, specify a Data Semantics function from it to \( \langle \text{SEM} \ I^*2 \rangle \), expressing our intuitive understanding of the Semantics of the Laws, and investigate its properties.

3) Schemas.

A much less general, less powerful and less comprehensive representations of an \( I^*2 \) or of a part thereof are schemas.

A schema specifies kinds of categories and relations in a database and properties of these categories and relations.
properties are of Integrity-imposing type, i.e. restricting the possibilities for \( SEM \ I^*1 \). Other properties are of
Inference-defining type, i.e. implying deduction of \( SEM \ I^*1 \) from
\( SYNT \ I^*1 \). (A property can be of both types.)
A schema itself is a Meta-data-base, i.e. a system of metafacts. Being a /*meta*/ data base, a schema can be like \( SYNT \ I^*1 \), or can be deduced like \( SEM \ I^*1 \) (the rules of deduction of a schema are imposed by the \( I^*3 \)). Respectively, there are two domains for schemas: a More Syntactic one, \( <SYNT \ I^*2> \), and a More Semantic one, \( <SYNT-SYNT-SEM \ I^*2> \).
As it was for \( I^*1 \), in Abstract Syntax Notation
\[
<SYNT \ I^*2> = <SYNT-SYNT-SEM \ I^*2> = \text{<System Of Facts>}
\]
/* Meta */ operations can be executed on schemas. We shall restrict them too (in this paper) to “schema-transactions” only. Schemas are investigated in section 3.5.

C. \( I^*3 \)

We shall specify and investigate only two different \( I^*3 \)'s. One will be that from which we shall extract the Semantics of Integrity and Inference Laws in section 3.5. The other will correspond to a Sample DBMS managing data bases according to Conceptual Binary schemas in which Inference and Integrity aspects of Categories and Relations are expressed. In section 3.5 the latter \( I^*3 \) will be represented Semantically, as a \( SEM \ I^*3 \), and “Syntactically”, as a \( SYNT \ I^*3 \), actually a Meta-schema.
The \( SYNT \ I^*3 \) is a system of (meta-meta-) facts, so
\[
<SYNT \ I^*3> = \text{<System Of Facts>}
\]
Other \( SYNT \ I^*3 \) (\( \langle SYNT \ I^*3 \rangle \)), with their semantics slightly different from that of the discussed \( SYNT \ I^*3 \), could be
1) Clarification of concepts.

Before formally defining the domain of Systems Of Facts, we would like to clarify several concepts.

a) An elementary fact states that a certain entity has a certain property, i.e., is of a certain category, or that a certain relation holds between two certain entities.

b) A category is a property. At every instant of time every category is assigned a set of entities, which are then of this category, i.e., have this property. Thus, a category should not be identified with its current set (Extension).

Example: "Paris is a City", but Paris € THE-SET-CORRESPONDING-NOW-TO-THE-CATEGORY( City ).

2) Representation of categories by relations.

A category can be represented by a binary relation as a set can be represented by a pairset.

Thus, we can simplify mathematical treatment of the model by dealing with Binary Relations only, while "Category" will be a particular type of Binary Relation. "Category" will be a (meta-) subcategory of the (meta-) Category "Relation", and will have certain (meta-) properties not common to the other Relations. Non-Semantic aspects will be "harmed" by this simplification.

9) The domain of Systems Of Facts.
A commented definition of the domain of Systems Of Facts in Abstract Syntax Notation follows.

There are two special elements in the domain: 1, denoting the Undefined System, and ERROR, denoting the invalid system. These are used mainly in <SEM T*1> to correspond to Data Bases from <SYNT T*1>, which are Illegal according to T*2. (If the DBMS is able to recognize the illegality and to treat it, then the corresponding <SEM T*1> is ERROR, if the DBMS would loop trying to check the legality or to recognize the illegality or to treat it, then the corresponding <SEM T*1> is !.). Additional meanings can also be assigned to these special elements.) Any other element of the domain is a Proper System Of Facts.

Every System Of Facts is represented by a finite mapping from a category can be represented by a binary relation. Note: for any set S, S=s"P"S; for any pairset P, PAIRS_SET REPRESENTING THE-SET(S)={ (x,x) | x ∈ S }
relations into a pairset. A relation, which is not mapped to any pairset is treated as being mapped to the empty pairset \( \emptyset \).

A system of facts \( P \models \{ R \mid \text{MAP} \} \) \( < \text{Pairset} P > \) \( \{ \text{MAP} \} \) \( < \text{Pairset} P > \)

A pairset is a set of pairs of entities \( \{ \text{Pairset} P \mid \text{SET} \} \) \( \{ \text{SET} \} \)

A relation is a metaentity \( \{ \text{Metaentity} P \} \) which is a particular case of entityification.

\[ \text{Relation } P = \text{Metaentity } P = \text{Entity } P \] \( \{ \ast \} \)

An entity can be represented by an internally-generated identifier or a value, which is an integer or a string. Internally identifiers and strings domain-bounded as represented by integers syntax notation follows.

The entity \( P \) is \( \{ \text{identifier } P \mid \text{value } P \} \).

 hứng the undervalued fact integer ERRORS stringing the invalid system. These are identifiers \( P \) which 

\( \{ \text{String } P \} \) which \( \text{integer } P \) 

if the DBMS is able to recognize the illegality and to treat it, then the concept \( P \) is either in the DBMS. 

All operators over the domain of systems of facts.

We shall need definitions of several operators and orders over the domain \( \{ \text{System of Facts} \} \). We should emphasize that there is 

\( \{ \ast \} \) - An equation of this type in Abstract Syntax Notation states that the left side domain is, or is isomorphic to, the set of all finite mappings (partial functions) from the first domain on the right side, to the second.

\( \{ \ast \} \) - This denotes the powerset or a set isomorphic to it.

\( \{ \ast \} \) - In Abstract Syntax Notation does not identify the connected concepts.

\( \{ \ast \} \) - Union of domains

\( \{ \ast \} \) - With such definition, the left domain is not equal to the right: its elements are distinguishable.
nothing in common between these mathematical operators (which we shall use for mathematical constructions) and database operations, e.g., transactions (treated in section 3.3), which refer to something executable in a database and have semantics.

a) The flat order "Less Defined Or Equal":

\[ \text{sf}_1 \leq \text{sf}_2 \ \text{iff} \ \text{sf}_1 \equiv \text{sf}_1 \lor \text{sf}_1 = \text{sf}_2 \]

(notes: \((\wedge), (\vee), (\cdot), (\equiv)\))

b) Naturally extended, [non-strict] inclusion predicate:

\[ \text{domain} \left( \text{sf}_1 \right) \subseteq \text{domain} \left( \text{sf}_2 \right) \rightarrow \left( \text{sf}_1 \left( \text{relation} \right) \subseteq \text{sf}_2 \left( \text{relation} \right) \right) \]

\((\equiv),(\ast)\)

c) Nonmonotonically extended, inclusion:

\[ \left( \text{sf}_1 \equiv \text{sf}_2 \right) \rightarrow \left( \text{domain} \left( \text{sf}_1 \right) \subseteq \text{domain} \left( \text{sf}_2 \right) \right) \]

\((\equiv), \left( \text{System Of Facts} \right) \rightarrow \left( \text{TRUE}, \text{FALSE}, '1', \text{ERROR} \right)\)

d) Naturally extended Union, Intersection and Subtraction of Systems Of Facts.

\[ \left( \text{domain} \left( \text{sf}_1 \right) \cup \text{domain} \left( \text{sf}_2 \right) \right) \rightarrow \left( \text{domain} \left( \text{sf}_1 \right) \cap \text{domain} \left( \text{sf}_2 \right) \right) \]

We shall need definitions of several operators and orders over the domain \(\left( \text{System Of Facts} \right)\). We should emphasize that there is

\((\wedge)\) 'A \wedge B' means 'A is defined to be B'.

\((\equiv)\) Lambda notation defines a function by specifying its value for every element of its domain, i.e., \(\lambda x. \exp = \{ (x, \exp) \mid \forall x \in \text{domain} \}\) for more information about lambda-calculus see [Stev-77a].

\((\equiv)\) \(\ast\) initiates the scope of a \(\wedge\) or of a logical quantor (\(\forall\)) or of a logical quantor (\(\forall\)) or of a logical quantor (\(\forall\)). The scope is closed by a parenthesis which was opened before the corresponding quantor (\(\forall\) or \(\exists\)) or by the end of the whole formula. In abstract syntax notation does not identify the concept of union of domains.

\((\equiv)\) With such a definition, the left domain is not equal to the right. It's elements are distinguishable.
Lemma.

(i) ' and ∪ are commutative and associative.

(ii) [', ∩ and ε are well-founded-orders.

(iii) ', ∪, ∩ and − are Total (Ω) and Computable (†).

(*) - Ω denotes the empty set.

(*∪) - union of sets (Naturally extended).

(*∩) - intersection of sets (Naturally extended).

(*−) - subtraction of sets (Naturally extended).

(Ω) - An operator * is Total iff

\[ (s_{f1}, s_{f2}) \in \text{System of Facts} \] \[ (s_{f1} * s_{f2}) = \text{if } s_{f1} = \text{ERROR or } s_{f2} = \text{ERROR, then } s_{f1} = \text{ERROR or } s_{f2} = \text{ERROR} \]

(†) - A mathematical function is Computable iff there exists a Program whose Denotational Semantics is this function.

(*) - The sign * closes the discussion of a Lemma, a theorem etc. In most cases we shall not present technical proofs in this paper.
3.3 SEMANTICS OF OPERATIONS.

The discussion of db operations will be restricted here to UPDATE TRANSACTIONS.

Transactions' semantics (intuitive and formal) is quite simple. Their semantics depend on the semantics of data (i.e., \( I \times r \) Data semantic of function implied by \( r \times r ^{2} \)) so the foregoing formal definition will be parameterized by it.

A. THE INTUITIVE SEMANTICS.

Transactions (atomic actions) Their execution is an indivisible process. Nevertheless, the change made in a data base by a Transaction needs not to be atomic. Consideration of a Transaction as an unbreakable unit updating possibly much information is motivated by current trends in data base management systems (*). This is important both for a multi-user environment and for integrity tests specifications in a single-user system, and \((\alpha)\) are Total (T) and compatible (C).

We shall consider Transactions which can be regarded as consisting of two systems of facts: those to be deleted from the data base, and those to be inserted into it. If after the completion of virtual deletion (first) and the insertion (second) the data base would lose its integrity, the Transaction is rejected and no update is done (alternatively, the db is restored to its previous status). No intermediate checks of integrity are done between or during the deletion and the insertion, because the db does not have to be 'integral' in the middle of the execution of an atomic operation. E.g. a

(*)- see e.g. [Adabas-78a].

(*I)- The sign \( \downarrow \) is used to denote semantic, etc. In most cases we shall not present formal
substitution of a value in a total function can consist of a deletion of the old value and an insertion of the new one (inbetween the function is not total and the db is not integral).

B. THE DOMAINS: SYNTACTIC AND SEMANTIC.

The following definition gives abstract representations of Transactions. Nevertheless, these are syntactic in comparison with standard semantic representations.

\[ \text{Transaction} : \text{<Deletion>, <Insertion>} \]

\[ \text{<Insertion> : <Deletion> = System Of Facts} \]

Output from execution of a transaction is either nothing, or an error message (if the Transaction tried to break integrity, but its execution was prevented), or is undefined (when the DBMS fails either to execute a Transaction or to reject it).

\[ \text{<output>} = \text{<nil> | ERROR} \]

(a)

The semantic domain of Transactions is standard:

\[ \text{<Semantics Of Transactions>} = \]

\[ \text{[<Transaction>, [<SYNT, f*1>],}} \]

\[ (<SYNT f*1> x <output>) \]

C. THE TRANSACTIONS SEMANTICS FUNCTION.

Let Ds be a data semantics function. The Semantics of Transactions function corresponding to it follows:

\[ \text{SEMANTICS-OF-TRANSACTIONS(Ds)} \]

\[ \text{<Transaction>} \]

\[ \text{LoadDb<SYNT f*1> : } \text{<output>} \]

(a) - The domain <output> has the flat order.
let Newdb = (Olddb -

(*let)

select <DELETE>(transaction)

select <INSERT>(transaction)

in if Ds (Newdb) = ERROR

then /* the Transaction tried to break integrity and 
   was rejected */

(/* the actual new db: */ Olddb,

/*the output: */ ERROR)

else (Newdb, nil).

Note: the basic operators used above are naturally extended, e.g.

(1 = x)\equiv

Lemma. \forall Ds \in [\langle \text{SYNT } \exists \ast \rangle \Rightarrow \langle \text{SEM } \exists \ast \rangle]. \text{SEMANTICS-OF-TRANSACTIONS}(Ds) \in 

[\langle \text{TRANSACTION} \Rightarrow [\langle \text{SYNT } \exists \ast \rangle \Rightarrow \langle \text{SEM } \exists \ast \rangle] \times \langle \text{OUTPUT} \rangle \rangle], \text{i.e.}

\text{cont.}\text{uous} \text{if } Ds \text{ is computable then } \text{SEMANTICS-OF-TRANSACTIONS}(Ds) \text{ is}

1\text{computable too.}

\text{Semantics of Transactions:}

\text{3.4 SEMANTICS OF GENERAL INTEGRITY AND INFERENCE LAWS.}

We shall now discuss the 'semantics' of a Data base whose behavior 

is controlled by a \\

information from which two Laws are 

derivable: the Law of Data Integrity and the law of information 

Inference. The Law of Data Integrity identifies valid states of 

Data base and prevents updates which can bring the Data base 

into an invalid state. The Law of information Inference 

\text{Olddb} \text{ } \text{LET } \exists \ast 

--------------

(*let)- 'let v=exp1 in exp2' is an abbreviation for 

\text{The}(\text{let } v=\text{exp1 in } \text{exp2}) \text{ has the same
identifies db states in which all possible deductions of information have been done. These deductions need not have been done actually or even virtually, but can just be comprehended by an external analyzer of the db contents. It will be a matter of Data semantics how to generate this logical state from a given state so that the Law would be satisfied. As the deducible information's validity should also be concerned, we state in the intuitive semantics that the Integrity Law should be applied to the deduced state. Practically, the Integrity and Inference Laws are somehow represented in the db schema, in defined exit procedures (specified to be automatically executed just before or just after db operations of certain types), etc., or can be interpretable by an advanced DBMS from some direct representations.

Thus, we shall neglect other aspects of I*2 and shall consider db driven solely by these Laws and I*3 information interpreting them.

Thus, we shall represent I*2 quasi-syntactically as a pair of predicates corresponding to the Laws -- relatively to the semantic representation, which is a standard SEM. I*2 as defined in section 2.6. A mapping between the representations will be defined as the I*2 Data semantics function DS20.

We call the aforementioned quasi-syntactic representation a SYNT-SEM-SEM I*2, distinguishing it from some more Syntactic representations discussed later.

The domain of SYNT-SEM-SEM I*2 representations (called <SYNT-SEM-SEM I*2>) is defined as follows:

/*Every SYNT-SEM-SEM I*2 is composed of an Integrity Law and an Inference law*/
<SYNT-SEM-SEM I*2>::<Integrity Law> <Inference law>

/*Every Law is either a total predicate over the domain of systems of facts (instantaneous Data bases) or is predefined as an error. (Non-total predicates also have interesting semantic aspects, but they are beyond the scope of this discussion.) */

<Integrity Law> = <Inference law> = <Law> =

(<System of Facts P> → (True | False)) | 1 | ERROR

The domain <Law> has the flat C.P.O. [L with bottom 1.

The order of the domain <SYNT-SEM-SEM I*2> is standard as defined in section 2.7 (*).

We define an I*2 Data semantics DS20 : [<SYNT-SEM-SEM I*2> + <SEM I*2>] as follows, (using the VDM meta language):

DS20 A

/* gives semantics for every pair of Laws. */

L(h,g)ε<SYNT-SEM-SEM I*2>,

/* h and g impose certain I*1 Data semantics... */

let Ds'(h,g) =

/* ... which, for every instantaneous db (a System of Facts) */

L sfacts ε <SYNT I*1>.

/*gives its semantics using the later defined standard transformation corresponding to the Inference Law. */

if /* the deduced logical db would not be valid with respect to the Integrity Law... */

not h(STANDARD-TRANSFORMATION(corr. to g)(sfacts))

/*note: this can give 1 (as h(1)=1),

--------------------------

(*)- If <A> :: <B> <C> then mk-<A>{b1,c1} is less defined or equal to mk-<A>{b2,c2} according to the standard order iff b1 is less defined or equal to b2 and c1 is less defined or equal to c2.
which propagates to the whole 'if' expression. /*
then /* the instantaneous db was meaningless */
ERROR
else /* the semantics of the instantaneous db is: */
  STANDARD-TRANSFORMATION(corresponds to g)(sfacts)
/* now, using this, */ in
/* the semantics of (h,g) is composed of... */
mk-<SEN I*2> ( /*the above defined I*1 Data semantics function */
  Ds'(h,g),
+ /* and the standard semantics of Transactions */
  DS20
+ /SEMANTICS-OF-TRANSACTIONS(Ds'(h,g)))
/* give */

We shall define now the standard transformation corresponding to an Inference Law g. Generally, different transformations can correspond to a Law, and the choice does not have to be Deterministic.

Definition. A transformation F:[<System Of Facts> -> <System Of Facts>] corresponds to an Inference Law g iff
/* for every System Of Facts... */
  *sf1e<System Of Facts P>.
  /* if it cannot be extended into a
     system satisfying g */
  if not [sf2e<System Of Facts P> . sf1=sf2 ∧ g(sf2)]
    /* then it is either transformed into the
        error state or is untransformable */
then \( P(sf1) \in \{1, \text{ERROR} \} \)

/* otherwise it is transformed to a minimal (with respect to \( c' \)) state satisfying \( g */^a 

else \( g(P(sf1)) \land \\
\\n\forall sf2 \in \langle \text{System Of Facts} P \rangle \text{ if} \\
g(sf2) \land sf1 \neq sf2 \Rightarrow P(sf1) \\
\text{then} sf2 = F(sf1) \quad \Box \\

We shall show later that for every Inference Law \( g \) there is at least one such transformation (which will be the standard transformation).

Of special interest are Deterministic Inference Laws, i.e. those which have unique transformation up to the treatment of unextendable instantaneous data bases (i.e. those for which no proper deduced data exists).

**Definition.** An Inference Law \( g \) is Deterministic iff

\[ \forall F1,F2:[\langle \text{System Of Facts} P \rangle \to \langle \text{System Of Facts} P \rangle], \]

if \( F1,F2 \) correspond to \( g \)

then \( \forall sf \in \langle \text{System Of Facts} P \rangle, P2(sf) \in \{F1(sf),1,\text{ERROR}\}. \quad \Box \\

For many practical cases, it is useful to know that intersection-closed Laws are Deterministic, as stated in the following theorem.

**Theorem.**

If \( \forall sf1,sf2 \in \langle \text{System Of Facts} P \rangle \),

\[ \text{if} \ g(sf1) \land g(sf2) \]
then \( g(\text{sf}1, \text{sf}2) \)

then \( g \) is Deterministic.

PROOF. (As an exception, a comprehensive proof is given for this theorem.)

1. Let \( g: \langle \text{System Of Facts} P \mapsto \{ \text{true, false} \} \rangle \be an Inference Law;

2. Assume that \( \forall \text{sf}1, \text{sf}2 \in \langle \text{System Of Facts} P \rangle, \text{ if } g(\text{sf}1) \land g(\text{sf}2) \)
then \( g(\text{sf}1, \text{sf}2) \);

3. Let \( F1, F2: [\langle \text{System Of Facts} T \mapsto \langle \text{System Of Facts} T \rangle \rangle \) be Transformations Corresponding to \( g \). (We have to prove that \( \forall \text{sf} \in \langle \text{System Of Facts} P \rangle; F2(\text{sf}) \in \{, \text{ERROR}, F1(\text{sf}) \} \));

4. Let \( \text{sf} \) be a System Of Facts;

5. Assume (to be contradicted) that \( F2(\text{sf}) \in \{, \text{ERROR}, F1(\text{sf}) \} \);

6. Ergo, \( F2(\text{sf}) \in \langle \text{System Of Facts} P \rangle; \)

7. Ergo, from definition of "corresponding transformation",

(7.1) \( \text{sf} \in F2(\text{sf}) \) \land

(7.2) \( g(F2(\text{sf})) \)

(7.3) \( \forall \text{sf} \in \langle \text{System Of Facts} P \rangle, \text{ if } \text{sf} \in F2(\text{sf}) \land g(\text{sf}) \)
then \( \text{sf} \in F2(\text{sf}) \);

(8) By definition of "corresponding transformation", from

(7.1) \land (7.2) follows:

\( F1(\text{sf}) \in \langle \text{System Of Facts} P \rangle \land g(F1(\text{sf})) \land \forall \text{sf} \in \langle \text{System Of Facts} P \rangle \)

\( \forall \text{sf} \in \langle \text{System Of Facts} P \rangle \land g(\text{sf}) \) \land \( \text{sf} \in F1(\text{sf}) \);

(9) Let \( \text{sf}3 := F1(\text{sf}) \land F2(\text{sf}) \);

(10) From (2), (7.2), (8) follows: \( \text{sf}2 \in \langle \text{System Of Facts} P \rangle \land g(\text{sf}) \)

(11) From (7.1), (8), (9) and properties of \( \land \) and \( \lor \), it follows:

\( \text{sf} \in F1(\text{sf}) \land \text{sf} \in F2(\text{sf}) \); \( \text{sf}3 = F2(\text{sf}) \);

(12) From (7.3), (10), (11) : \( \text{sf}3 = F2(\text{sf}) \);

(13) From (8), (10), (11) : \( \text{sf}3 = F1(\text{sf}) \);

(14) From (12), (13) : \( F2(\text{sf}) = F1(\text{sf}) \), in contradiction to (5);

(16) Ergo, \( F2(\text{sf}) \in \{, \text{ERROR}, F1(\text{sf}) \} \);

(17) From ((3) \Rightarrow (4) \Rightarrow (16)) and by definition of Determinism, \( g \)
is Deterministic.

\( \square \)

Not only strictly Deterministic Laws are of practical interest. For example, a Law implying existence of an entity under certain conditions, can be obeyed/Implemented by generating a new internal identifier and connecting it in relations. The choice of such a new identifier is not deterministic, but is not relevant from the semantic point of view. Some standard procedure can be chosen to generate it. Although, in other cases, a Law can be even less Deterministic intuitively, a standard procedure can be chosen to implement it, thus
then \( g(\text{sf1}\cdot \text{sf2}) \).

**Proof.** (As an exception, a comprehensive proof is given for this theorem.)

1. Let \( g: \text{System Of Facts P} \rightarrow \{\text{true, false}\} \) be an Inference Law.
2. Assume that \( \text{sf1, sf2} \in \text{System Of Facts P} \). If \( g(\text{sf1}) \land g(\text{sf2}) \) then \( g(\text{sf1}\cdot \text{sf2}) \).
3. Let \( F1, F2: [\text{System Of Facts} \rightarrow \text{System Of Facts}] \) be Transformations Corresponding to \( g \). (We have to prove that \( \text{sf1, sf2} \in \text{System Of Facts P}, F2(\text{sf}) \in \{1, \text{ERROR}, F1(\text{sf})\} \).
4. Let \( sf \) be a System Of Facts.
5. Assume \( \{1, \text{ERROR}, F1(\text{sf})\} \).
6. Ergo, \( F2(\text{sf}) \in \text{System Of Facts P} \).
7. Ergo, from definition of "corresponding transformation".
8. By definition of ""corresponding transformation", from (7.1) and (7.2) follows:
   \[ F1(\text{sf}) \in \text{System Of Facts P}, \land g(F1(\text{sf}) \land \text{sf} \in F1(\text{sf}) \land \text{sf} \in \text{System Of Facts P} \).
   If \( \text{sf} \in F1(\text{sf}) \land g(\text{sf}) \) then \( \text{sf} \in F2(\text{sf}) \).
9. Let \( \text{sf3} = F1(\text{sf}) \cdot \text{sf2} \).
10. From (2), (7.2), (8) follows: \( \text{sf2} \in \text{System Of Facts P} \) \land g(\text{sf3})
11. From (7.1), (8), (9) and properties of \( \cdot \) and \( \lor \), it follows:
12. From (7.3), (10), (11) follows: \( \text{sf3} \in F2(\text{sf}) \).
13. From (8), (10), (11) \( \text{sf3} \in F4(\text{sf}) \).
14. From (12), (13) \( F2(\text{sf}) = F1(\text{sf}) \), in contradiction to (5).
15. Ergo, \( F2(\text{sf}) \in \{1, \text{ERROR}, F1(\text{sf})\} \).
16. \( \text{sf} \in \text{System Of Facts P} \). \( \text{sf} \in \text{System Of Facts P} \).
17. From ((13) \rightarrow (4) \rightarrow (16)) and by definition of Determinism, \( g \) is Deterministic.

Not only strictly Deterministic Laws are of practical interest. For example, a Law implying existence of an entity under certain conditions, can be obeyed/implemented by generating a new internal identifier and connecting it in relations. The choice of such a new identifier is not deterministic, but is not relevant from the semantic point of view. Some standard procedure can be chosen to generate it. Although, in other cases, a Law can be even less Deterministic intuitively, a standard procedure can be chosen to implement it, thus
Determinizing it.

In order to define the standard transformation, we need the following lemma.

**Lemma.** There exists an enumeration \((\text{first}<\text{System Of Facts } P>, \text{next}<\text{System Of Facts } P>\rightarrow<\text{System Of Facts } P>\)\) preserving \(\prec\), i.e. \((\forall \text{sf}<\text{System Of Facts } P>. \exists i<\text{Natural(incl.0)} P>. \text{sf}=(\text{next}^{\ast}i)(\text{first})) \land \forall \text{sf}<\text{System Of Facts } P>. \forall i<\text{Natural(excl.0)} P>. (\text{next}^{\ast}i)(\text{sf}) \prec \text{sf}\).

**Sketch of proof:** \(<\text{System Of Facts } P>\) is enumerable; \(\prec\) is a well-founded order. The result follows from the theory of enumeration of well-founded sets. \(\square\)

Let \((\text{first}0, \text{next}0)\) be such an enumeration. (The standard enumeration.) The definition of \text{STANDARD-TRANSFORMATION} follows.

**Definition.** Let \(g:<\text{System Of Facts } P>\rightarrow\{\text{true, false}\}\) be an Inference Law.

\[\text{STANDARD-TRANSFORMATION(corr. to } g) \triangleq \forall \text{sf}1<\text{System Of Facts}>.\]

\[\text{let } F' := /* \text{function "computing" the deduction of } sf1\]

starting from a given System Of Facts \(sf2\) */

(Least Fixed Point (\(\exists\)) with respect to the [\

pointwise(\(\prec\)) order, of the following functional:

\[
\forall F<[<\text{System Of Facts}>\rightarrow<\text{System Of Facts}>].
\]

\[(1\mapsto1, \text{ERROR} \mapsto \text{ERROR}) \cup \]

\[\forall \text{sf}2<\text{System Of Facts } P>.\]

\[\text{if } \text{sf}1 \prec \text{sf}2 \land g(\text{sf}2) \text{ then } \text{sf}2\]

\[\text{else } F(\text{next}0(\text{sf}2))\]

\[\text{in } F'(\text{sf}1)\]

\(\square\)

**Lemma.** \text{STANDARD-TRANSFORMATION(corr. to } g) \in[<\text{System Of Facts } P>].\]
Theorem. STANDARD-TRANSFORMATION(corr. to g) is a Transformation Corresponding To g. □

Theorem. For any Deterministic Inference Law g, STANDARD-TRANSFORMATION(corr. to g) does not depend on the choice of standard enumeration, and is Minimal among the transformations -- with respect to the pointwise order [.

Sketch of proof. From the definition of the STANDARD-TRANSFORMATION, it is provable that any System Of Facts is mapped by any STANDARD-TRANSFORMATION into its deductive extension (which is unique for the Deterministic law g according to the definition of determinism) iff the System Of Facts is extendable, otherwise it is mapped to i. Assuming that there are two STANDARD-TRANSFORMATIONS for different enumerations, it follows from the above that they must be equal. Also, any other Corresponding transformation must give the same result for the deductively-extendable Systems Of Facts (due to the Determinism). As the other elements are mapped by the STANDARD-TRANSFORMATION to i, it is less defined than, or equal to, any other Corresponding transformation. In a precise proof, special attention is paid to transformations that map some elements to ERROR. □

The STANDARD-TRANSFORMATION does not have to be computable, but:

Theorem. If g is computable then STANDARD-TRANSFORMATION(corr. to g) is computable, i.e. there exists a program whose mathematical semantics is STANDARD-TRANSFORMATION(corr. to g).

Sketch of proof:

The definition of STANDARD-TRANSFORMATION(corr. to g) is convertible to a recursive program that uses an external functional routine computing g (g is total and computable). □

(2) F' is the Least Fixed Point of a functional T iff (F'=T(F')) ∧ ~F".if F'=T(F") then F'[F"].

(+) (Pointwise order with respect to [-])Δ

Δ(F',F")<the-domain-of-functions**2.

~sf∈domain(F'). F'(sf)['F"(sf)
A rather simple DBMS, managing any data base according to metafacts from its schema, will be studied in the following.

A 'schema', or a SYNT I*2, is a system of metafacts about the relations and the categories that are defined in the data base.

Users can update the schema of their data base, thus altering the I*2 represented by the schema.

A Deduced Schema, or a SYNT-SYNT-SEM I*3, is a system of all the facts which are deducible from a given schema.

Analogically to SEM I*1, a deduced schema is supposed to be comprehended by users and by the DBMS, but it does not have to be stored anyhow, or to be directly retrievable. Whether or not it is directly retrievable depends on the semantics of schema retrieval operations, which is beyond the scope of this discussion.

Schemas themselves, being instantaneous meta data bases, are schematized by a meta-schema, which is a SYNT I*3.

We shall consider a certain I*3, which we shall represent syntactically as a metaschema, and semantically as a SEM I*3.

The definition of a SYNT I*3 follows.
We shall now define the $SEM \ast 3$ corresponding to the above metaschema, thus semanticizing it.

```
sem31 \( \Delta \) mk-<SEM \( \ast 3 \)> (  
  /* the data semantics function which we have to define */  
  DS2,  
  /* the semantics of metatransactions altering schema, which we have to define */  
  SEMANTICS-OF-SCHEMA-TRANSACTIONS)
```

DS2 is the composition of three semantics functions:

(i) \( \text{DS22}: \langle \text{SYNT I}\ast 2 \rangle \rightarrow \langle \text{SYNT-SYNT-SEM I}\ast 2 \rangle \), transforming a Valid Schema into a Deduced Schema;

(ii) \( \text{DS21}: \langle \text{SYNT-SYNT-SEM I}\ast 2 \rangle \rightarrow \langle \text{SYNT-SEM-SEM I}\ast 2 \rangle \), transforming a Deduced Schema into a pair of Laws (Integrity and Inference);

(iii) \( \text{DS20}: \langle \text{SYNT-SEM-SEM I}\ast 2 \rangle \rightarrow \langle \text{SEM I}\ast 2 \rangle \), transforming a pair of Laws into a comprehensive semantic representation. (DS20 has been defined in section 3.4.)

In order to define the above, we need some preliminary definitions.

First, we define the following total predicate, stating for every pair of pairsets whether they hold a given relational meta-property. For unary properties, only the first pairset is tested whether it is of this meta-category.

**Definition.** THE-PAIR-OF-PAIRSETS-HAS-PROPERTY \( \Delta \)

\[ \forall (r,r2)\in\langle \text{Pairset}\rangle\ast 2, \text{prop}\in\text{metaschema1(Relation)}. \]

\[ \text{select prop:}(*), \]

\[ \text{Relation } \mapsto \text{TRUE}, \]

\[ \text{Category } \mapsto r=p"(s"r) \]

\[ \text{Integrity-property } \mapsto \text{TRUE} /*\text{any Relation may be defined as an Integrity-property*}/, \]

\[ \]

\[ (*) - \text{In Abstract Syntax Notation "select" is an abbreviation for a mathematical function which is similar to the Denotation of "select" in programming languages. We emphasize that our constructs are not programs and are not Operational semantics, but are mathematical functions.} \]

\[ \]
Inference-property $\mapsto$ TRUE,
Inverse $\mapsto$ $r=r^2*r^{-1}$,
Binary-relation $\mapsto$ TRUE,
Function $\mapsto$ $\forall (x,y),(z,u) \in r$. if $x=z$ then $y=u$,
Reflexive $\mapsto$ $\forall(x,y) \in r. (x,x),(y,y) \in r$,
Irreflexive $\mapsto$ $\forall(x,y) \in r. x \neq y$,
Symmetric $\mapsto$ $r=r^*{-1}$,
Antisymmetric $\mapsto$ $\forall(x,y) \in r. (y,x) \neq r$,
Transitive $\mapsto$ $r \circ r \circ r$,
Equivalence $\mapsto$ $r=r^*{-1} \land r\circ r^*{-1}=r \land r\circ r=r$,
Strict-partial-order $\mapsto$ $r \circ r \neq r \land r^n r^*{-1}=\emptyset$,
Subcategory $\mapsto$ $r \subset r^2$,
Disjoint $\mapsto$ $r \cap r^2=\emptyset$,
From $\mapsto$ $\forall(x,y) \in r. x \in s^n r^2$,
From-total $\mapsto$ $\text{domain}(r) \cap s^n r^2$.

Lemma. The above predicate is total and computable. □

Now, for every Deduced Schema and a set of metaproperties to be tested, we shall define a predicate over the domain of Systems of Facts to test whether these properties hold for all the pairsets of the System of Facts according to the specification in the Deduced Schema.

Definition. \text{PPF} \triangleq L \text{ schema} \triangleq \text{SYNT-SYNT-SEM} \ I^*2 \ L \text{ set-of-meta-properties} \triangleq \text{POWERSET}(s^n \text{metaschema('Relation')}).

if schema\in\{,\text{ERROR}\} then /*propagate error*/ schema else $Lsf\in\text{System Of Facts} P$. 
let sf' := /* as sf, but the non-represented empty relations are now explicit */
sf ∈ Relation ∈ schema('Relation'). (/*the empty pairset:*/ %)
in
(∀ metavariable ∈ set-of-meta-properties. ∀(rel1, rel2) ∈ schema(metaveriable).
THE-PAIR-OF-PAIRSET-HAS-PROPERTY(
    sf'(rel1), sf'(rel2), metavariable)
∧ /* the special semantics for the metavariable 'Relation'
*/
if 'Relation' ∈ set-of-meta-properties
    then /* all the relations, which are assigned pairsets', i.e. */ domain(sf) /* are defined in the schema, i.e. */
    ∈ schema('Relation')

Lemma. ∀schema ∈ <SYNT-SYNT-SEM> I*2>-{1, ERROR}.
∀set-of-meta-properties ∈ POWERSET( s"metaschema1('Relation')").
PFP(schema)(set-of-meta-properties) is Total and Computable ∧
(∀sf1, sf2 ∈ System Of Facts P).
    if PFP(schema)(set-of-meta-properties)(sf1) ∧
PFP(schema)(set-of-meta-properties)(sf2)
    then PFP(schema)(set-of-meta-properties)(sf1n'sf2)),
and thus PFP(schema)(set-of-meta-properties) is Deterministic. □

Now we can define DS21:
DS21A \( L \text{schema} < \text{SYNT-SYNT-SEM I*2} > \\
\text{mk-<SYNT-SEM-SEM I*2>(} \\
\text{select-<Integrity-Law>:} \\
PFP(\text{schema})(s"\text{metaschema1(Integrity-property)}") \\
\text{select-<Inference-Law>:} \\
PFP(\text{schema})(s"\text{metaschema1(Inference-property)}") \\
\)

As to DS22, it is actually the schema deduction transformation. I*2 is deduced according to the metaschema in the same way as I*1 information is deduced according to a schema. Thus, we define:

DS22A \( \text{select-<I*1 Data Semantics Function> (DS20(DS21(metaschema1)))} \)

/* note: metaschema1 is already fully deduced from itself*/

The definition of a complete I*2 data semantics function follows:

\[ \Psi^2 A DS22 \circ DS21 \circ DS20 \]

Lemma. \( DS2 \in [<\text{SYNT I*2} \rightarrow <\text{SEM I*2}>] \) (i.e. DS2 is Continuous) and \( \forall \text{schema} \langle \text{SYNT I*2} \rangle DS2(\text{schema}) \) is Computable.

Sketch of proof: By the definitions, previous lemmata and the results of section 3.4. []

It remains to define the semantics of transactions on schemas.
As, up to this, a schema can be regarded as a /*meta*/ data base, the semantics of /*meta*/ transactions on schemas is the standard semantics of transactions with respect to *I* data semantics:

**SEMANTICS-OF-SCHEMA-TRANSACTIONS**

**SEMANTICS-OF-TRANSACTIONS(DS22)**

Now we have completely defined sem31.

**Quasi-theorem.** sem31 matches our intuitive semantics of the /*simplified*/ Conceptual Binary DBMS having the given metaschema.

**Theorem.** sem31 ∈ <SEM I*3> and it is Computable in the sense that there exists a program (DBMS), whose Denotational Semantics is sem31.

**Sketch-of-proof:** The result follows from the computability results in the previous lemmas. □

**Conclusion:**

We have presented a new approach to defining formal semantics of data bases. We have applied this approach to formal semantization of some important aspects of data bases, generally and of Conceptual Binary data base specifically. These included data integrity, data deduction, semantic schemas, non-procedural definition of data base behavior and other aspects. This paper did not intend to define full semantics of any DBMS.
ACKNOWLEDGMENT

I am very grateful to Nissim Francez for advising me and helping me very much in this work.

I am very grateful to Jacob Raz, Makowski, Jehoshua Shokhat and Anna Kissin

for their valuable remarks.

REFERENCES.

[Adiba-78a] M. Adiba, "Modelling Approach for Distributed Data Bases", in [Bracchi-78a].
[Falkenberg-78a] E.D. Falkenberg. "Data Models: The Next Five Years", in [Infotech-78b].
[Bracchi-78a]
[Steel-77a] T.B. Steel "The current ANSI/ X3/ SPARC/ DMBS Proposals", in [Infotech-78b].
[Vandijck-77a] E. Vandijck, "An Overview of Current Relational Data Base Query Languages", in [Infotech-78b].