Formal syntax and semantics of a reconstructed relational database system

Dan Jonsson
Dept. of Sociology, Göteborg University,
Brogatan 4, S-413 01 Göteborg, Sweden

1. *Introduction*

In formal logic, the definition and analysis of logical systems frequently proceeds as outlined below:

(1) A non-empty set of (formal) *languages* is specified. Each such language may be identified with a non-empty set of *well-formed formulas* (wffs). A language L may be specified by listing (i) a set of *logical symbols* (e.g., $\neg$, $\wedge$, $\vee$), (ii) a set (*"vocabulary"*) V of *non-logical symbols*, and (iii) a set of *formation rules* showing how to form wffs from primitive (logical and non-logical) symbols and/or other wffs. Formal languages are often analyzed within some context of fixed sets of logical symbols and formation rules, so that attention is confined to some set of languages each of which corresponds to a particular vocabulary of non-logical symbols.

(2) Given a formal language L over a vocabulary V, a distinguished subset *Syn* of L is specified. For example, *Syn* may be defined recursively as the smallest set of wffs containing (a) a finite set of stated *axioms* and (b) all wffs that may be deduced from wffs in *Syn* through the use of certain stated *derivation rules*.

(3) A non-empty set I of (formal) *interpretations* of (all wffs in) L is specified. Each wff in L may be characterized as being either true or not true under any particular interpretation in I.

(4) A subset *Int* of I is specified. A distinguished subset *Sem* of L is then specified with reference to the formal interpretations in *Int*. For example, *Sem* may be defined as the set of wffs which are true under *all* interpretations in *Int*.

(5) The distinguished subsets *Syn* and *Sem* are compared. (One frequently wants to show that *Syn* = *Sem*.)

In this article, systems of database states will be analyzed in the spirit of steps (1) – (5) above. The analysis is based on the notion that although database states are not formulas in the strict sense – i.e., finite sequences of symbols – they may nevertheless be treated as are well-formed formulas in formal logic. As a consequence, the formal syntax and semantics of systems of database states may be analyzed in much the same way as the formal syntax and semantics of systems of well-formed formulas.

The database system defined and discussed here is a reconstructed relational system. This article also introduces a more formal way of looking at Icaros-type semantic models as presented in [4]. Specifically, the database schemas to be defined below, and their graphical representations in particular, correspond to Icaros-type models.

2. *Databases*

In this section, database schemas and database states consistent with database schemas will be defined. In the definitions below, let S^* (S^+) denote the set of all (non-empty) subsets of S. As usual, $f(S) = \{f(t) \mid t \in S\}$. Finally, '$f : X \Rightarrow Y$' should be read '$f : X \rightarrow Y$ and $f(X) = Y$'.

Definition 1. A *database schema* is a tuple $V = \langle K, I, Z_\omega, Z_\pi, A, R, \kappa, \zeta_i, \zeta_a, \rho \rangle$ satisfying the four conditions listed below.

(1) K, I, Z_ω, Z_π, A, and R are non-empty, pairwise disjoint sets, whose elements are referred to as *values* (or *constants*), *intensions*, *object slots*, *predicate slots*, *attributes* (or *aspects*), and *relations* (or *relation nodes*), respectively.

Let Z denote $Z_\omega \cup Z_\pi$.

(2) κ, ζ_i, ζ_a, and ρ are functions such that $\kappa : I \Rightarrow K$, $\zeta_i : I \Rightarrow Z$, $\zeta_a : A \Rightarrow Z$, and $\rho : A \Rightarrow R$.

An intension i is said to *connect* k *and* z (symbolized $k\hat{}i\hat{}z$) iff $\kappa(i) = k$ and $\zeta_i(i) = z$, and an attribute a is said to *connect* z *and* r (symbolized $z\hat{}a\hat{}r$) iff $\rho(a) = r$ and $\zeta_a(a) = z$.

(3) No slot is connected to some value by two distinct intensions. (Values connected to object slots serve as *identifiers*, and values connected to predicate slots serve as *characterizers*; cf [4].)

(4) Every relation is connected to at least one object slot and at least one predicate slot. (Attributes connecting relations and object slots are called *identification attributes*, while attributes connecting relations and predicate slots are called *characterization attributes*.) ◊

Definition 2. Alternatively, a database schema may be defined as a tuple $\langle K, Z_\omega, Z_\pi, A, R, Dom, \zeta_a, \rho \rangle$ satisfying the three conditions listed below.

(1) K, Z_ω, Z_π, A, and R are non-empty, pairwise disjoint sets.

(2) Dom, ζ_a and ρ are functions such that $Dom : Z \rightarrow K^+$, $\cup_{z \in Z} Dom(z) = K$, $\zeta_a : A \Rightarrow Z$, and $\rho : A \Rightarrow R$. (Sets of the form $Dom(z)$ for some $z \in Z$ are called *domains*).

(3) Every relation is connected to at least one object slot and at least one predicate slot ◊

Proposition 1. Definitions 1 and 2 are equivalent. That is, for each $V = \langle K, I, Z_\omega, Z_\pi, A, R, \kappa, \zeta_i, \zeta_a, \rho \rangle$ satisfying (1) – (4) in Definition 1 one can define a function Dom such that $V' = \langle K, Z_\omega, Z_\pi, A, R, Dom, \zeta_a, \rho \rangle$ satisfies (1) – (3) in Definition 2, and for each V' satisfying (1) – (3) in Definition 2 one can similarly define I, κ, and ζ_i so that (1) – (4) in Definition 1 are satisfied.

Proof sketch. Given V, define $Dom : Z \rightarrow K^+$ by $Dom(z) = \{k \mid k\hat{}i\hat{}z \text{ for some } i \in I\}$. Given V', define $I = \{(k, z) \mid z \in Z \text{ and } k \in Dom(z)\}$, $\kappa((k, z)) = k$, and $\zeta_i((k, z)) = z$. ◊

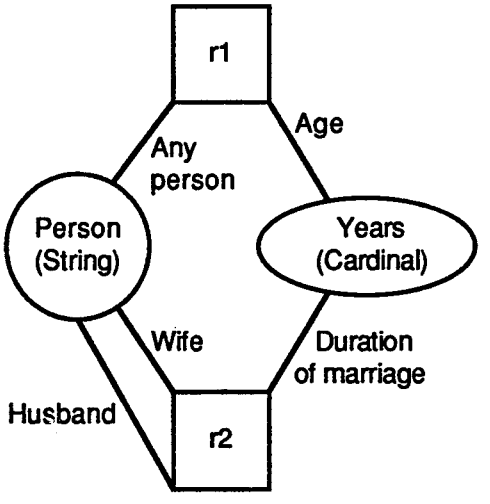
Note that according to Definition 1, a database schema may be regarded as a graph with nodes $R \cup Z_\omega \cup Z_\pi \cup K$, edges $A \cup I$, source function $\rho \cup \zeta_i$ (say), and target function $\zeta_a \cup \kappa$ (say). Definition 1 is more appealing from a formal point of view, but Definition 2 is closer to current database theory, so it is the one that will be used below.

Example 1. A reconstructed relational database schema

K = String ∪ Cardinal (predefined sets);
Z_ω = {Person}, Z_π = {Years};
A = {Any person, Husband, Wife,
Age, Duration of marriage};
R = {r1, r2};
Dom(Person) = String, Dom(Years) = Cardinal;
ζ_a(Any person) = Person, ζ_a(Husband) = Person,
ζ_a(Wife) = Person, ζ_a(Age) = Years,
ζ_a(Duration of marriage) = Years;
ρ(Any person) = {r1}, ρ(Husband) = {r2}, ρ(Wife) = {r2},
ρ(Age) = {r1}, ρ(Duration of marriage) = {r2};

A convenient and compact graphical representation of the schema in Example 1 is shown below. In this representation, relations are shown as rectangles, object slots as circles, predicate slots as ellipses, and attributes as edges connecting rectangles with circles or ellipses. Rectangles, circles, ellipses, and edges are labeled with names of the coresponding relations, object slots, predicate slots, and attributes, respectively, while domains associated with object slots or predicate slots are shown within parentheses in the corresponding circles or ellipses.

Example 2. An ICAROS-type graphical model for the database schema defined above



As a third option, the schema can be described by means of pseudo-DDL statements.

Example 3. Pseudo-DDL statements for the database schema defined above

```
ObjectSlot Person is String;
PredicateSlot Years is Cardinal;
Relation r1 is Any person, Age;
        Attribute Any person is Person;
        Attribute Age is Years;
Relation r2 is Husband, Wife, Duration of marriage;
        Attribute Husband is Person;
        Attribute Wife is Person;
        Attribute Duration of marriage is Years;
```

Definition 3. Given a database schema $V = \langle K, Z_\omega, Z_\pi, A, R, Dom, \zeta_a, \rho \rangle$, let $Alloc_\omega : R \rightarrow A^+$ be the function given by $Alloc_\omega(r) = \{a \mid z\hat{}a\hat{}r \text{ for some } z \in Z_\omega\}$. Similarly, $Alloc_\pi : R \rightarrow A^+$ is the function given by $Alloc_\pi(r) = \{a \mid z\hat{}a\hat{}r \text{ for some } z \in Z_\pi\}$. Let furthermore Γ_ω be the function defined on R given by $\Gamma_\omega(r) = \{\gamma_\omega \mid \gamma_\omega : Alloc_\omega(r) \rightarrow K$, where $\gamma_\omega(a) \in Dom(\zeta_a(a))$ for every $a \in Alloc_\omega(r)\}$, and let Γ_π be the function defined on R given by $\Gamma_\pi(r) = \{\gamma_\pi \mid \gamma_\pi : Alloc_\pi(r) \rightarrow K$, where $\gamma_\pi(a) \in Dom(\zeta_a(a))$ for every $a \in Alloc_\pi(r)\}$.

A *tuple* consistent with r for some $r \in R$ is a pair $(\gamma_\omega, \gamma_\pi) \in \Gamma_\omega(r) \times \Gamma_\pi(r)$. A *database state* consistent with V is a function S defined on R such that $S(r)$ is a finite subset of $\Gamma_\omega(r) \times \Gamma_\pi(r)$ for every $r \in R$. The set of all database states consistent with V constitutes the *state space* spanned by V. ◊

Note that the concepts of database schema, database state, and state space spanned by a particular database schema are analogous to the formal logic notions of vocabulary of non-logical symbols, well-formed formula, and language over a particular vocabulary, respectively.

Database states may be represented in the usual tabular form. (Note, however, that in the present case slots as well as attributes are shown in the heading of each relation table.)

Example 4. A database state consistent with the database schema defined above

r1

Any person Person	Age Years
Peter	36
Mary	36

r2

Husband Person	Wife Person	Duration of marriage Years
Peter	Mary	3

A database state presupposes a database schema, and it is sometimes essential to consider database states in conjunction with corresponding database schemas.

Definition 4. A *database (instance)* is a pair $\langle V, S \rangle$, where V is a database schema and S is a database state consistent with V. ◊

To relate the present reconstruction of relational databases to previous notions, note that each of the terms *domain* and *attribute* is somewhat ambiguous in conventional relational database theory. First, "domain" may refer to a *data type* (e.g., "string") as well as a *"semantic type"* (e.g., "supplier name") [1]. Second, "attribute" may refer to an *attribute name* as well as an *attribute occurence.* (The latter distinction is necessary in conventional database theory, since columns – i.e., attribute ocurrences – in different relations may have the same name – i.e., attribute name. In queries, update operations etc, attribute occurences are usually identified by relation name / attribute name combinations.)

Also note that semantic types and attribute names are overlapping notions: they both serve to express semantic connections between (attribute ocurrences in) different relations, and either notion may be used for this purpose. (That is, semantic interconnections can be expressed by letting relations share either attribute names or domains regarded as semantic types.) In the present approach, object and predicate slots are shared between relations and express semantic interconnections.

Table 1. Conventional notions and corresponding notions in the present approach

Conventional notion	Present notion
Attribute (occurence) Relation_name.attribute_name	Attribute
(Semantical) domain Attribute (name)	Slot
(Data) domain	Domain

Note that in the present approach the notions of attribute name and attribute occurence coincide. (Expressed in an equivalent way, attribute names are globally unique.) Note, also, that whereas the notion of intention is redundant in the sense that every intention is identified by a unique slot-value pair, the situation is different for attributes, as shown by Examples 1–4 above.

Finally, the distinction between identification attributes and characterization attributes in the present approach is analogous to the distinction between key and non-key attributes (respectively) in the conventional approach. There is an important difference, though, in that keys are independent of interpretations of databases – i.e., they are formal or syntactic in nature – whereas identification and characterization attributes have specific semantic significances, as will become apparent below.

In view of what has just been said, it is possible to approximate the database schema in Example 3 using a Data Definition Language supporting keys and domains.

Example 5. An approximate rendering of the database schema in Example 3

```
Domain Person is String;
Domain Years is Cardinal;
Relation r1 is Any person, Age;
        Key is Any person;
        Attribute Any person is Person;
        Attribute Age is Years;
Relation r2 is Husband, Wife, Duration of Marriage;
        Key is Husband, Wife;
        Attribute Husband is Person;
        Attribute Wife is Person;
        Attribute Duration of Marriage is Years;
```

3. *Functional normal form*

Definition 5. Given a database instance $\langle V, S\rangle$, a relation $r \in R$ is said to be in *functional normal form* (FNF) iff for any $(\gamma_{1\omega}, \gamma_{1\pi}), (\gamma_{2\omega}, \gamma_{2\pi}) \in S(r)$ it holds that $\gamma_{1\omega} = \gamma_{2\omega}$ implies $\gamma_{1\pi} = \gamma_{2\pi}$. $\langle V, S\rangle$ is said to be in FNF iff every $r \in R$ is in FNF. ◊

In other words, $\langle V, S\rangle$ is in FNF iff for each relation the set of all identification attributes constitutes a key – though not necessarily a minimal key, or a unique key.

Functional normal form is conceptually simple, then, but that does not mean that it is simplistic or naive. While it is beyond the scope of this article to compare FNF to traditional normal forms, I shall indicate the main thrust of the argument. FNF is based on the notion that a distinction should be made between "data-structuring" vs. "incidental" [3] or "constitutive" vs. "qualifying" dependencies [4], the former type of dependency being more relevant for database design than the latter. In the present approach, specifically, only functional dependencies according to which a characterization attribute is functionally dependent on the full set of identification attributes are regarded as data-structuring, and only such dependencies are taken into account when deciding whether or not a database state is in FNF. (Note that the designation of certain functional dependencies as data-structuring presupposes and is relative to the designation of certain slots as identification slots. The implications of this 'relativity principle' cannot be pursued here, however.)

Incidentally, if an FNF relation is subject to no other constraints (apart from domain constraints) than those implied by its being in FNF, then the identification attributes constitute a candidate key (i.e., a minimal key) and the relation is in Fagin's [2] domain-key normal form (DKNF). Also, a relation in DKNF is in FNF provided that the set of identification attributes coincides with the set of key attributes. Loosely speaking, FNF is equivalent to DKNF as far as data-structuring functional dependencies as defined above are concerned.

4. *Interpretations*

Definition 6. Given a database schema V, a tuple $\langle O, F_\omega, F_\pi\rangle$ satisfying conditions (1) – (3) below constitutes an *interpretation* of any database instance of the form $\langle V, S\rangle$.

(1) O is a non-empty set of *"objects"*.

(2) F_ω is a function defined on R such that $F_\omega(r)$ is a function defined on $Alloc_\omega(r)$ such that $F_\omega(r)(a) : Dom(\zeta_a(a)) \rightarrow O$.

Let Q be the function defined on R given by $Q(r) = \{\phi \mid \phi : Alloc_\omega(r) \rightarrow O\}$. (The elements of $Q(r)$ are *"object constellations"* associated with r.)

(3) F_π is a function defined on R such that $F_\pi(r)$ is a function defined on $Alloc_\pi(r)$ such that $F_\pi(r)(a) : Dom(\zeta_a(a)) \rightarrow Q(r)^*$. ◊

Definition 7. (i) Given F_ω, let $\mathbb{F}_\omega$ be the function defined on R such that $\mathbb{F}_\omega(r) : \Gamma_\omega(r) \rightarrow \mathcal{Q}(r)$ is the function defined by the condition $\phi = \mathbb{F}_\omega(r)(\gamma_\omega)$ iff $\phi(a) = F_\omega(r)(a)(\gamma_\omega(a))$ for every $a \in Alloc_\omega(r)$. (ii) Given F_π, let $\mathbb{F}_\pi$ be the function defined on R such that $\mathbb{F}_\pi(r) : \Gamma_\pi(r) \rightarrow \mathcal{Q}(r)^*$ is the function defined by the condition $\phi \in \mathbb{F}_\pi(r)(\gamma_\pi)$ iff $\phi(a) \in F_\pi(r)(a)(\gamma_\pi(a))$ for every $a \in Alloc_\pi(r)$. (That is, $\mathbb{F}_\pi(r)(\gamma_\pi) = \bigcap_{a \in Alloc_\pi(r)} F_\pi(r)(a)(\gamma_\pi(a))$.) ◊

One will normally assume that $F_\omega(r)(\Gamma_\omega(r)) \supseteq F_\pi(r)(\Gamma_\pi(r))$ for every $r \in R$. This means that every object constellation that can be characterized by available characterizers can also be identified by available identifiers. As a consequence, tuples will not contain 'null values' associated with identification attributes. (Cf. 'entity integrity' as defined in [1].)

On the other hand, one could also assume that $F_\pi(r)(\Gamma_\pi(r)) \supseteq F_\omega(r)(\Gamma_\omega(r))$ for every $r \in R$ (or equivalently, $F_\pi(r)(a)(Dom(\zeta_a(a))) \supseteq F_\omega(r)(\Gamma_\omega(r))$ for every $r \in R$ and every $a \in Alloc_\pi(r)$). This would mean that every object constellation that can be identified by available identifiers can also be characterized by available characterizers. As a consequence, tuples will not contain 'null values' associated with characterization attributes.

Definition 8. Let $I = \langle O, F_\omega, F_\pi \rangle$ be an interpretation of $\langle V, S \rangle$. S is said to be *true* in I iff $F_\omega(r)(\gamma_\omega) \in F_\pi(r)(\gamma_\pi)$ for every $(\gamma_\omega, \gamma_\pi) \in S(r)$ for every $r \in R$. ◊

5. *Discriminating interpretations*

To be able to characterize database instances from a semantical point of view, notions of discriminating interpretations will be defined.

Definition 9. F_ω is said to be discriminating iff $k_1 \neq k_2$ implies $F_\omega(r)(a)(k_1) \neq F_\omega(r)(a)(k_2)$ for every $r \in R$ and every $a \in Alloc_\omega(r)$. Similarly, F_π is said to be discriminating iff $k_1 \neq k_2$ implies $F_\pi(r)(a)(k_1) \cap F_\pi(r)(a)(k_2) = \varnothing$ for every $r \in R$ and every $a \in Alloc_\pi(r)$.

An interpretation $\langle O, F_\omega, F_\pi \rangle$ is said to be *ω-discriminating*, *π-discriminating*, and *discriminating* iff, respectively, F_ω, F_π, and each of F_ω and F_π are discriminating. ◊

Intuitively, an ω-discriminating interpretation is one where distinct identifiers refer to distinct objects. In a π-discriminating interpretation, distinct characterizers refer to disjoint sets of object constellations. If in addition $F_\pi(r)(a)(Dom(\zeta_a(a))) = F_\omega(r)(\Gamma_\omega(r))$, then the characterizers in $Dom(\zeta_a(a))$ provide an exhaustive classification into non-overlapping categories of all object constellations in $F_\omega(r)(\Gamma_\omega(r))$.

6. *Syntax vs. semantics*

Speaking in the terms used in the Introduction, we have now shown how to define *V*, we have defined *L*, given *V*, and we have defined *Syn*, *I*, and *Int*, given *L*. We shall finally define Sem_1 and Sem_2 and show that $Syn = Sem_1 = Sem_2$. (Note that in this case, Sem_1 and Sem_2 are defined as sets of wffs which are true in *at least one* interpretation in *Int* rather than all such interpretations.)

Proposition 2. If $D = \langle V, S \rangle$ is in FNF, then there is some discriminating interpretation I of D such that S is true in I.

Proof sketch. Let $D = \langle V, S \rangle$ be any database instance, and set $I = \langle K, F_\omega, F_\pi \rangle$, where (i) $F_\omega(r)(a)(k) = k$ for every $r \in R$, $a \in Alloc_\omega(r)$ and $k \in Dom(\zeta_a(a))$, and (ii) $F_\pi(r)(a)(k) = \{\gamma_\omega \mid (\gamma_\omega, \gamma_\pi) \in S(r)$ and $\gamma_\pi(a) = k\}$ for every $r \in R$, $a \in Alloc_\pi(r)$ and $k \in Dom(\zeta_a(a))$. The following claims are easily verified: (a) I is an interpretation of D; (b) F_ω is discriminating; (c) if $\langle V, S \rangle$ is in FNF then F_π is discriminating; (d) $F_\omega(r)(\gamma_\omega) = \gamma_\omega$ for every $r \in R$ and $\gamma_\omega \in \Gamma_\omega(r)$; (e) $F_\pi(r)(\gamma_\pi) = \{\gamma_\omega \mid (\gamma_\omega, \gamma_\pi) \in S(r)\}$ for every $r \in R$ and $\gamma_\pi \in \Gamma_\pi(r)$. Now, if $(\gamma_\omega, \gamma_\pi) \in S(r)$ then $\gamma_\omega \in \{\gamma_\omega \mid (\gamma_\omega, \gamma_\pi) \in S(r)\}$. That is, $F_\omega(r)(\gamma_\omega) \in F_\pi(r)(\gamma_\pi)$ for every $r \in R$ and every $(\gamma_\omega, \gamma_\pi) \in S(r)$. Hence, S is true in I. ◊

Proposition 3. If there is some π-discriminating interpretation $I = \langle O, F_\omega, F_\pi \rangle$ of $D = \langle V, S \rangle$ such that S is true in I, then D is in FNF.

Proof. Assume that S is not in FNF, so that there is some $a_0 \in Alloc_\pi(r)$ for some $r \in R$ such that $(\gamma_\omega, \gamma_{1_\pi})$, $(\gamma_\omega, \gamma_{2_\pi}) \in S(r)$ and $\gamma_{1_\pi}(a_0) \neq \gamma_{2_\pi}(a_0)$. Assuming that S is true in I, $F_\omega(r)(\gamma_\omega) \in F_\pi(r)(\gamma_{1_\pi})$ and $F_\omega(r)(\gamma_\omega) \in F_\pi(r)(\gamma_{2_\pi})$, so $F_\pi(r)(\gamma_{1_\pi}) \cap F_\pi(r)(\gamma_{2_\pi}) \neq \varnothing$. By Definition 7, $F_\pi(r)(\gamma_{1_\pi}) \subseteq F_\pi(r)(a_0)(\gamma_{1_\pi}(a_0))$ and $F_\pi(r)(\gamma_{2_\pi}) \subseteq F_\pi(r)(a_0)(\gamma_{2_\pi}(a_0))$. Hence, $F_\pi(r)(a_0)(\gamma_{1_\pi}(a_0)) \cap F_\pi(r)(a_0)(\gamma_{2_\pi}(a_0)) \neq \varnothing$ although $\gamma_{1_\pi}(a_0) \neq \gamma_{2_\pi}(a_0)$, so F_π is not discriminating. Therefore, if S is not in FNF then it is not the case that S is true in some π-discriminating interpretation of $\langle V, S \rangle$. This proves Proposition 3. ◊

Proposition 4. The following three conditions are equivalent: (a) D is in FNF; (b) D possesses a true discriminating interpretation; (c) D possesses a true π-discriminating interpretation.

Proof. (a) implies (b) according to Proposition 2, (b) obviously implies (c), and (c) implies (a) according to Proposition 3; hence, all three conditions are equivalent. ◊

7. *Conclusions and research directions*

The idea of regarding a database state as a wff in a formal language whose syntax and semantics is to be specified leads naturally to a reconstruction of relational database theory when combined with two other ideas:

(i) For reasons sketched above, the attribute-domain dichotomy in standard relational database theory should be replaced by an attribute-slot-domain trichotomy.

(ii) In standard relational database theory, a tuple $t = v_1 \ldots v_m$ in relation r is seen as representing the formula $r(v_1 \ldots v_m)$ in predicate logic. From the present point of view, however, t represents an n-predicate formula $v_{i_1} \ldots v_{i_n}(v_{i_{n+1}} \ldots v_{i_m})$ in a generalized predicate logic. (Note that $v_{i_1} \ldots v_{i_n}$ are values associated with predicate slots, while $v_{i_{n+1}} \ldots v_{i_m}$ are values associated with object slots.)

From these points of departure, the analysis leads to the conclusion that there is a close connection between (a) database schemas, (b) semantical / conceptual models, and (c) formal logical systems related to predicate calculus. To be specific, we have seen that there is a close analogy between database schemas in a reconstructed relational system and vocabularies of non-logical symbols in certain logical systems, and it has also been shown that Icaros-type semantical models [4] provide direct graphical representations of these database schemas.

Note that the database schemas defined in this article correspond to vocabularies that contain only constant symbols and that database states (or instances) may be regarded as conjunctions of ground atomic formulas. Negation, disjunction, variables, and quantifiers should also be possible to accomodate within a similar framework, however.

A desirable property of a database state is that it is true under some interpretation. This property, as we have seen above, is related to a normal form based on functional dependencies. Another desirable feature is that distinct database states have distinct meanings as defined in terms of formal interpretations. It is conjectured that this feature is related to normal forms based on multivalued and join dependencies.

It was noted in connection with Definition 1 that database schemas may be regarded as graphs, in fact acyclic graphs. Interestingly, it seems possible to accomodate nested relations and complex objects in the present approach by allowing cycles involving attributes, and also to accomodate function symbols by allowing cycles involving intentions. Also, I have shown previously [4] that notions of object type and inheritance can be defined in terms of Icaros-type models.

References

[1] Date, CJ. An Introduction to Database Systems, Vol. 1, 5th Ed., Addison-Wesley 1990.

[2] Fagin, F. A Normal Form for Relational Databases That is Based on Domains and Keys. ACM TODS, Vol. 6, No. 3, Sept 1981.

[3] Fagin, F., A.O. Mendelzon & J.D. Ullman. A Simplified Universal Relation Assumption. ACM TODS, Vol. 7, No. 3, Sept 1982.

[4] Jonsson, D. Semantic Modeling Through Identification and Characterization of Objects. SIGMOD RECORD, Vol. 19, No. 1, March 1990.