1986

On derivation of network and relational schemas from an enhanced conceptual structure

Leszek A. Maciaszek
*University of Wollongong*

William S. Bowie
*University of Wollongong, uow@bowie.edu.au*

Stephen K. Lucas
*University of Wollongong*

---

**Recommended Citation**


*Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au*
ON DERIVATION OF NETWORK AND RELATIONAL SCHEMAS FROM AN ENHANCED CONCEPTUAL STRUCTURE

Leszek A. Maciaszek, William S. Bowie*, Stephen K. Lucas
University of Wollongong
Department of Computing Science
P.O.Box 1144, Wollongong, NSW-2500
Australia

ABSTRACT

The derivation rules from an enhanced conceptual structure into a logical schema for network and relational databases are examined. The initial conceptual structure (being itself a derivation from the predefined set of user requirements on data) captures the extended range of semantic modeling constructs, such as: partial and total entities of a relationship, weak and regular relationships, ternary, recursive and nonunivocal relationships, subset and generic relationships. An enhanced conceptual structure is subjected to transforms into logical schemas in a process which is tractable by computer-assisted tools and guarantees to produce feasible network and relational structures. Ten of the basic conceptual structures - from the overall number of one hundred transforms recognizable by the system - are presented with respect to the network database design and then again for the relational database design. The CAD-tool, written in Macintosh Pascal, is briefly described.

Categories and Subject Descriptors: H.2.1 [Database Management]: Logical Design; H.2.4 [Database Management]: Systems.

Additional Keywords and Phrases: Conceptualization, Conceptual Structure, Logical Design, Logical Schema, Network and Relational Model, Entity, Relationship, Record, Set, Relation, Non-Key Attribute, Primary Key, Foreign Key.

This work was supported in part by the University of Wollongong under Grant 03/143/407.

* Current address: Victoria Personal Computers Ltd., 511-620 View Street, Victoria, B.C. V8W 1J6, Canada.
Motto:

"The tools can give the database designer facts that are otherwise unavailable, but much of the process is still an art and relies heavily upon the designer's skill."

Teorey and Fry, 1982

1. INTRODUCTION

There is a general consensus that the major database models are equivalent (Biller and Neuhold, 1977; Borkin, 1978; Brady, 1985; Chen and Kuck, 1984; Imielinski and Lipski, 1982; Jajodia et al., 1983; Katz and Wong, 1982; Kerschberg et al., 1977; Kuck and Sagiv, 1982; Lum et al., 1985; Michaels et al., 1976; Navathe and Cheng, 1983; Schek, 1985; Subieta, 1985; Tsur and Zaniolo, 1984; Wong and Katz, 1980). Hence, it is tempting to work out tools for automatic conversion of database structures between different models (as well as between different levels of structures within the same model). Indeed, many attempts to this end have been reported (Batory, 1984; Borgida et al., 1982; Carlis and March, 1983; Codd, 1979; Dahl and Bubenko, 1982; De Antonellis and Di Leva, 1985; Gerritsen, 1975; Hawryszkiewycz, 1985; Hubbard, 1979; Hubbard, 1981; Navathe and Cheng, 1983; Roussopoulos, 1979; Wong and Katz, 1980; etc.).

However, with the notable exception of a few such attempts (e.g. DATAID as intermodel (Comp, 1985) and RM/T as intramodel approach (Codd, 1979)) most proposals are either oversimplified to be applicable or theoretical and practical components of them are not proportionally weighted.

We describe an alternative set of rules to transform an extended conceptual database structure into a pertinent logical schema for network as well as relational models. Following a tradition that grows out of semiotics we differentiate among syntactic, semantic, and pragmatic aspects of the transforms (Carberry et al., 1979; Subieta, 1985). This differentiation is well rooted in the various levels of syntactic, semantic, and pragmatic support of the three models under consideration (extended entity-relationship, network, relational). We rank (in the range 1 (the best) through 3 (the worst)) the syntactic, semantic, and pragmatic power of the models as shown in Figure 1.

<table>
<thead>
<tr>
<th>Models</th>
<th>Syntactics</th>
<th>Semantics</th>
<th>Pragmatics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Conceptual</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Network</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Relational</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 1 Relative Attractiveness of Database Models.

A major difficulty in handling conversions between database structures arises from different support of the "input" and "output" structures in the three discussed aspects (and especially in semantics). The (sub)structures that do not exhibit deep differences at this point can be converted automatically, often in a straightforward 1:1 mapping process. The other structures require a significant interaction on part of the designer or even a user of the subject area. Some conversions
can only be tuned after feedbacks from the successive design stages, in particular those dealing with performance prediction. In general, the derivation rules must ensure input/output equivalence or - to put it another way - must ensure information preserving transforms; that is, the transforms merely change the information format, not its semantics (note again the paramount importance of semantics in the modeling process).

We believe that a contribution of this report is that in our transforms we clearly address the three modeling aspects. Another contribution accrues from the wide extensions of our conceptual structures as compared with the standard entity-relationship model due to Chen (1976). Moreover, the transforms have been tractable by computer-assisted design tools that guarantee to produce feasible network and relational schemas for any conceptual structure. Finally, our initial conceptual structure is very attractive practically as it can be by itself derived in a computer-aided fashion from the user requirements on data. In fact, this report is an outgrowth of a wider, integrated database design methodology (Korczak and Maciaszek, 1979; Maciaszek, 1981; Maciaszek, 1982; Maciaszek, 1983; Maciaszek, 1984; Maciaszek, 1985a; Maciaszek, 1985b; Maciaszek, 1986).

2. TERMINOLOGY AND CLASSIFICATION

For completeness sake, we first describe our understanding of semiotics in database modeling. **Syntactics** deals with the relation between names and their formal interpretation; it involves only matters concerning the arrangement of symbols in a structure (e.g. from the fact that the two structures are interpreted as being equivalent in a 1:1 fashion we can conclude that the objects of the two structures can bear the same names). **Semantics** determines the relation between names and what they refer to or what they mean; it refers to the notion of truth in a structure (e.g. semantics of an attribute COLOR is different for the paint factory as compared with, for instance, a car manufacturer). **Pragmatics** determines the relation between the constructs of the system and their users. It can be considered as a measure of user friendliness and efficiency of the system; therefore, it is concerned with the psychological and practical aspects of the system. Studying the practical aspect of pragmatics presupposes that the system has been implemented.

A usual way of presenting the psychological aspect of pragmatics is through examples - and in this sense the pragmatics expresses the special case semantics. To introduce the general case semantics one needs to comprehend a full set of examples, that is if S denotes the derivation system and C denotes the set of all possible conceptual substructures, then the semantics of S transforms is given by the semantics function \( \text{sem}: S \otimes C \rightarrow L \); where \( L \) is the set of logical substructures for the network or relational model (see Subieta (1985) for more discussion on semantics and pragmatics).

The basic terminology related to the **conceptual database model** follows (cp. Maciaszek (1983), Maciaszek (1984), Maciaszek (1986)).

An **object** is a real or an abstract component of the information system as it is perceived by humans. In our methodology it merely serves as a common term for an entity and a relationship. An **entity** is a fundamental "thing" ("anything") of interest to an enterprise which can distinctly be recognized, identified and described (e.g. STUDENT, COURSE). A **relationship** reflects an interdependence or interaction among entities (e.g. ENROLLMENT), among entities and other relationships (e.g. PROJECT between the entity SUPERVISOR and the relationship JOB), or within an entity among its attributes (e.g. PREREQUISITES).

A property or characteristic of an object is called an **attribute** \( A \) and can be expressed as a function from an object \( Y \) into an associated **value set** \( V_A \). An attribute \( A \) may be compound (**group attribute**) and then is associated with a domain \( \text{Dom} = \text{Dom}_1 \otimes \ldots \otimes \text{Dom}_k \) (\( k \geq 2 \)) and range \( \text{Ran} \).
Ran_1 \otimes \ldots \otimes Ran_k (k \geq 2)$ or it may be multiple-valued, possibly compound, $(repeating\ attribute)$ and then is associated with a domain $Dom = Dom_1 \otimes \ldots \otimes Dom_k (k \geq 1)$ and range $Ran = 2Ran_1 \otimes \ldots \otimes Ran_k (k \geq 1)$.

We distinguish between the entity kind $E$ (often called entity set) and the entity instance $e$ (also entity or entity occurrence), the relationship kind (set) $X$ and the relationship instance $x$ (relationship or relationship occurrence). Instances of an attribute kind $A$ are called, ut supra, values $v_A$.

An outcome of the conceptualization process is then the set of entity kinds $E = \{E\}$, the set of relationship kinds $X = \{X\}$ such that $X = \{E_1, \ldots, E_m, X_1, \ldots, X_n\}$, $m+n \geq 1$ is the degree of $X$, and the set attribute kinds $A = \{A\}$ such that the propositions $\forall E \exists A [A \in E], \forall A \exists E \exists X [(A \in E) \lor (A \in X)]$ and the predicate $\exists X [A \in X]$ hold.

Relationship kinds are classified according to four different criteria: (1) optionality, (2) identification, (3) degree, and (4) complexity.

With regard to the optionality criterion, the participation of the objects in a relationship can be total or partial. A total object of the relationship kind (denoted $Y_X$(total)) means that every instance $y$ of an object kind $Y$ involved in the relationship kind $X$ occurs in a certain instance $x$ of that relationship kind. The objects of the relationship kind which do not satisfy this condition are called partial $Y_X$(partial). If all objects of the relationship kind are total, then -by virtue of a shorthand- the relationship kind is called total $X$(total). Accordingly, a relationship kind is partial $X$(partial) if all objects of it are partial. More formally:

$Y_X$(total) = $\forall y, y \in X, y$ is an instance of $Y [y \vDash x] x$ is an instance of $X$]

$Y_X$(partial) = $\exists y, y \in X, y$ is an instance of $Y [y \not\vDash x] x$ is some instance of $X$]

$X$(total) = $(\forall Y, Y$ is an object kind of $X) (\forall y, y$ is an instance of $Y)$

$[y \vDash x] x$ is an instance of $X]$

$X$(partial) = $(\exists Y, Y$ is an object kind of $X) (\exists y, y$ is an instance of $Y)$

$[y \not\vDash x] x$ is some instance of $X]$

A division by criterion of identification results in regular and weak relationship kinds. We say that a relationship kind is regular $X$(regular) when it is identified by its own attributes; otherwise -if a full identification of the relationship kind requires concatenation of attributes from the participating objects - the relationship kind is said to be weak $X$(weak). In first-order logic:

$X$(regular) = $(\exists K, K$ is a key of $X) (\forall A, A \in K) [A \in X]$

$X$(weak) = $(\exists K, K$ is a key of $X) (\forall A, A \in K) [A \in X]$

A criterion of degree refers to the total number of objects linked by a relationship kind. From this point of view one can distinguish sole relationship kinds $X$(sole) (involving one entity kind only and expressing recursive dependencies among the values of attributes of that single entity kind) from ample relationship kinds $X$(ample) (associating two or more object types). The ample relationship kinds are usually subject to further classification by degree which results in binary (degree two) and $n$-ary (three or more) relationship kinds. In our notation:

$X$(sole) = $\exists P, E \vDash P, X = E = \{A\} = [(e_i \otimes v_P) \rightarrow (e_j \otimes v_P)] \rightarrow x$
According to the criterion of complexity we separate out singular and multiple relationship kinds. The multiple relationship kinds can be divided further into univocal and nonunivocal relationship kinds. A relationship kind is singular when among its object instances the only possible linkings are 1:1. Multiple relationship kinds may be of 1:N or N:1, and then the "subordination" between objects is univocal, or they may be of M:N, and in that case the "subordination" is not univocal. To put it formally:

\[ X(\text{singular}) = \{Y\}, \text{card}(\{Y\}) \geq 1 \Rightarrow \left( \exists y_i \exists y_j \ (y_i \not\equiv y_j) \rightarrow x \right) \]

\[ i \neq j, y_i \text{ is an instance of } Y_i, y_j \text{ is an instance of } Y_j, x \text{ is an instance of } X \]

We will now proceed with the terminology of the logical network model in the scope needed by our derivation system (Draft, 1985a; Maciaszek, 1983).

A record is the logically distinct ordered collection of data-items (e.g. STUDENT). It is the basic unit of manipulation in the Network Database Language NDL (Draft, 1985a). A set expresses the interdependence between its component records that must be maintained by the DBMS (e.g. PREREQUISITES). A set can also be perceived as the collection of records. Each set has one record type designated as its owner and one or more record types designated as its members.

A data-item (or item) \( D \) is the generic unit of logical data structure and can be seen as a function from a record \( R \) into an associated value set \( \text{val}(D) \). A data-item \( D \) may be compound (group item) and then it is associated with a domain \( \text{Dom} = \text{Dom}_1 \otimes \ldots \otimes \text{Dom}_k \) (\( k \geq 2 \)) and range \( \text{Ran} = \text{Ran}_1 \otimes \ldots \otimes \text{Ran}_k \) (\( k \geq 2 \)) or it may be multiple-valued, possibly compound, (repeating item) and then is associated with a domain \( \text{Dom} = \text{Dom}_1 \otimes \ldots \otimes \text{Dom}_k \) (\( k \geq 1 \)) and range \( \text{Ran} = 2^{\text{Ran}_1} \otimes \ldots \otimes \text{Ran}_k \) (\( k \geq 1 \)). (In Draft (1985a) group and repeating items are called arrays of data-items.)

We distinguish between the record type \( R \) and the record occurrence (or record) \( r \). In a sense, a record type defines a collection of record occurrences and integrity constraints that these occurrences must satisfy. We also distinguish between the set type \( S \) and the set occurrence (set) \( s \). Occurrences of a data-item type \( D \) are called, as indicated above, values \( \text{val}(D) \).

A logical schema of network database is defined - leaving aside the integrity rules - in terms of the following sets: the set of record types \( R = \{R\} \); the set of set types (i.e. the mathematical set of network set types) \( S = \{S\} \), such that \( S = \{R_1, \ldots, R_m\} \), \( m \geq 1 \); and the set of data-item types \( D = \{D\} \) such that the propositions \( \forall R \exists D [D \in R], \forall S \forall D [D \in S], \forall D \exists R [D \in R] \) hold.

For comparison purposes, sets can be classified according to the four criteria used for relationships classification: (1) optionality, (2) identification, (3) degree, and (4) complexity.

The optionality criterion fits well for the sets. A set type can be total or partial. By analogy, we can define: a total record type of the set type \( R_S(\text{total}) \), a partial record type of the set type
RS\textsubscript{total}, a total set type \( S\textsubscript{total} \), and a partial set type \( S\textsubscript{partial} \). (To be precise, for member record types we should differentiate further between the optionality of insertion and retention. The former divides set types into: (a) automatic, (b) manual, (c) structural, whereas the latter into: (a) fixed, (b) mandatory, (c) optional.)

\[
\begin{align*}
RS\textsubscript{total} &= \forall r, r \in S, s \text{ an occurrence of } R \quad [r \in s \mid s \text{ an occurrence of } S] \\
RS\textsubscript{partial} &= \exists r, r \in S, s \text{ an occurrence of } R \quad [r \notin s \mid s \text{ some occurrence of } S] \\
S\textsubscript{total} &= (\forall R, R \text{ a record type of } S) \quad (\forall r, r \text{ an occurrence of } R) \\
\quad [r \in s \mid s \text{ an occurrence of } S] \\
S\textsubscript{partial} &= (\exists R, R \text{ a record type of } S) \quad (\exists r, r \text{ an occurrence of } R) \\
\quad [r \notin s \mid s \text{ some occurrence of } S]
\end{align*}
\]

The classification by identification is not useful for the set types. At best, one can claim that all set types are weak. The point is that the sets do not have existence on their own in the database - they are only the collections of records and on this basis they inherit the properties of the records involved.

The criterion of degree is again adaptable to the set types (as defined in the standard proposal (Draft, 1985a) and implemented in at least one production DBMS - Cullinet's IDMS). The set types can be \textit{sole} (recursive) \( S\textsubscript{sole} \) or \textit{ample} \( S\textsubscript{ample} \). The ample set types can be further divided into \textit{binary} and \textit{n-ary}, under the condition that the set type has exactly one owner record type.

\[
\begin{align*}
S\textsubscript{sole} &= \exists P, P \supseteq P, S = R = \{D\} = \{((r_i \otimes v_p) \rightarrow (r_j \otimes v_p)) \rightarrow s \mid i \neq j, r_i, r_j \text{ are occurrences of } R, s \text{ an occurrence of } S]. \\
S\textsubscript{ample} &= \{R\}, \text{card}(\{R\}) > 1 = [\exists r_i \exists r_j \quad (r_i \otimes r_j ) \rightarrow s \mid i \neq j, i = 1, j = 1, ..., n, r_i \text{ is an owner occurrence of } R_i, \\
r_j \text{ are member occurrences of } R_j, s \text{ an occurrence of } S]
\end{align*}
\]

Finally, the criterion of complexity is also applicable to set types. The set types can be \textit{singular} \( S\textsubscript{sing} \) or \textit{multiple}. However, the multiple set type can only be \textit{univocal} \( S\textsubscript{mult-univ} \), in accordance with the rule that "a member record occurrence of a member (record type) of a given (set type) is a member record of at most one set of that (set type)" (Draft, 1985a). (Note, that a singular set type, as widely known in a network database community - that is the set type having SYSTEM as its owner record type - is, with respect to this classification, the univocal set type.) A potentially nonunivocal set type can be easily represented by two or more univocal set types.

\[
\begin{align*}
S\textsubscript{sing} &= \{R\}, \text{card}(\{R\}) \geq 1 = [\forall s \forall r \quad (\text{card}(r_o) = 1) \land \text{((\text{card}(r_m) = 0) \lor \text{((\text{card}(r_m) = 1))}) \mid r_o \text{ is the owner of } s, r_m \text{ are the members of } s] \\
S\textsubscript{mult-univ} &= \{R\}, \text{card}(\{R\}) \geq 1 = [\forall s \forall r \quad \text{card}(r_o) = 1) \land \text{((\text{card}(r_m) = 0) \lor \text{((\text{card}(r_m) = 1))}) \mid r_o \text{ is the owner of } s, r_m \text{ are the members of } s]
\end{align*}
\]

Finally, we describe the terminology relevant to the \textit{logical relational model} (Draft, 1985b; Maier, 1983) - again in the scope required by our derivation rules. (Incidentally, talking about \textit{logical relational model} is superfluous in that the relational model is concerned with the logical level of the system, not the physical level.)
A table (relation) T is the multi-set of rows (e.g. STUDENTS, PREREQUISITES) (v. Knuth (1969) for the definition of multi-set, also called the bag). A row (n-tuple) N is the non-empty sequence of values of columns belonging to the table. A column (attribute) C is the multi-set of values that may vary over time. Corresponding to each column is a set D, called the domain of C. (The concept of domain is still a controversial research topic, yet it is very desirable for practical database purposes. To quote a passing comment made recently by Codd (1986): "Other people (often the vendors themselves) dismiss the domain concept as "academic". My reply to them is: The atom bomb was also academic.") Every column of a relation is unique and simple, i.e. neither group nor repeating columns are permitted. The atomic column restriction is called the first normal form. (Another recent citation at this juncture: "The first-normal-form constraint does not limit the power of the relational approach, but it does limit its naturalness." (Korth, 1986).) A column has an ordinal position within a table.

We distinguish between the table definition (relation scheme) T and a table (relation, tabulation) t. The smallest unit of data that can be selected from (and/or updated in) a table is a value of a column vC. Therefore, a row is in fact a set of pairs (Ci : Vi(C)), i = 1, ..., n and n is the degree of a table definition. The row is the smallest unit of data that can be inserted into a table or deleted from a table. The number of rows in a table is called its cardinality. Since the current relational systems do not support the concept of domain, the definition of a column includes only its data type and an indication whether the column can contain null values.

A logical schema of relational database consists of the set of base table definitions B = {B}, the set of view table definitions V = {V}, and the set of columns C = {C}. (A view definition defines a view table derived directly or indirectly from one or more other tables by the evaluation of relational algebra operations. In other words, the view table is the table that would result if the relational operations were executed. However, not all view tables are updatable (Date, 1986; Draft, 1985b, Merrett, 1984).

Regretably, we are unable to pursue our previous classification for the relational model - mainly because of semantic limitations of this model (Figure 1). Instead, we briefly introduce the notions of primary and foreign key, as being indispensable to cope with some of the derivation strategies.

The user's (or preferably and desirably - the system's) knowledge of primary and foreign keys is absolutely fundamental to the operation of the overall relational model. As pointed out by Date (1986): "Foreign-to-primary-key matches represent references from one relation to another; they are the "glue" that holds the database together.". In other words, the foreign-to-primary-key matches express relationship kinds (in the conceptual sense) between tables, or - more precisely - between rows of tables.

A key (candidate key, designated key) K is basically a unique identifier of table rows. If T is a table definition with columns C1, C2, ..., Cn then a key K is a subset of T (T ⊇ K) such that for any distinct rows n1 and n2 in any tabulation t, n1(K) ≠ n2(K) (uniqueness property) and no proper subset K' of K (K ⊆ K') shares this property (minimality property). It follows that: (1) the key can be simple or compound (i.e. multicolumn), (2) the table may have more than one key and then we choose one of the keys to be the primary key PK and the remainder are called the alternate keys. A foreign key FK is the key in one table definition, say T1, such that its set of values {vFK} in a tabulation t1 of T1 is the same as the set of values of the primary key {vPK} of a tabulation t2 of some table definition T2. Note that {vFK} = {vPK} should be drawn from a common domain D.
3. DIAGRAMMATIC NOTATION

As emphasized, the methodology has been tractable by CAD tools. Therefore, it relies heavily upon graphical facilities provided within the framework of software development aids - in our case: Macintosh Pascal, QuickDraw and ToolBox. The facilities are not restrictive. To the contrary, they are powerful enough to inspire the implementor in many respects. In a nutshell, the Macintosh has a mouse, a high quality graphics library (QuickDraw) easily accessed from within Pascal programs, and a ToolBox of routines for the construction of pull-down menus and handling external events such as dragging the mouse (which allows shapes to be moved around on the screen). We now proceed with presenting the graphical counterparts for the conceptual and logical notions defined in Section 2.

The graphical representation for conceptual structure is shown in Figure 2 (v. Maciaszek (1986)). The ovals stand for relationship kinds, the rectangles - for entity kinds. The lines join objects involved by a relationship kind. The cardinality of lines coming out of a relationship oval shows its degree. Variations in shading of the relationship oval are used to denote weak or regular relationship kinds. The white ovals represent weak, the gray - regular relationship kinds. Variations in darkness of the gray shade are caused by existence of two different sole regular relationship kinds: homogeneous and heterogeneous. In general, we divide sole relationship kinds into three categories: (a) hierarchical recursion, (b) homogeneous network recursion, (c) heterogeneous network recursion. Those categories are exemplified in Figure 3 (v. Maciaszek (1986)).

![Figure 2. Diagrammatic Notation for an Enhanced Conceptual Structure.](image)

The homogeneous and heterogeneous network recursions differ in the way of implementing the concept of role (Bachman, 1977; Hawryszkiewycz, 1984). Roles allow the DBA to treat entity instances from the same entity kind in different ways. Depending on the role taken by the entity instance in the relationship instance, the entity instances may differ in some attributes (e.g. the attribute DISEASE is only applicable to PERSON assuming the role PATIENT). This sort of nonuniform treatment of entity instances is inherent in the heterogeneous network recursion. The facilitating factor of the homogeneous case is that attributes of the entity instances are always the same and do not depend on the role taken by the entity instances.
It is evident that the differences between the homogeneous and heterogeneous cases should be mirrored in a diagram of the conceptual structure. Therefore, we introduced the dark-gray ovals to represent the relationship kinds in the heterogeneous network recursion. The underlying meaning is that those relationship kinds are always made regular (to avoid the possible ambiguities of semantic interpretation, if not for other reasons). The relationship kinds in the homogeneous network recursion and in the hierarchical recursion are permitted to be either weak (white ovals) or regular (light-gray ovals). However, one can readily distinguish between the hierarchical and network case because the former is always univocal, whereas the latter - nonunivocal (see the following paragraph).

Figure 3. The Diagrams of Sole Relationship Kinds.

Simple lines are used to indicate the singular relationship kind. That is, if all the lines connecting the relationship oval with its objects are simple, then the relationship is singular. If at least one of the lines has a semicircle attached to it, then the relationship is multiple. More specifically, if only one line ends in semicircle the multiple relationship is univocal, otherwise - nonunivocal (the use of semicircles instead of arrows is motivated by purely technical reason connected with the graphics library of Macintosh Pascal). A line (with or without semicircles) can be thick or thin. A thin line means that the object is partial in the relationship kind at hand. And vice versa, a thick line specifies the total object of the relevant relationship kind.
At present, our CAD package does not handle unambiguously the nested relationship kinds (i.e. relationships of relationships). However, we have worked out a consistent graphical convention to picture nested relationship kinds (v. Figure 11 in Maciaszek (1986)) and currently we are extending appropriately our CAD conceptualization programs. For the time being, we assume (without loss of quality) that the input for our transforms from conceptual structure will not include nested relationship kinds (v. Heuristic 9 in Maciaszek (1986)).

The graphical notation for logical network structure is presented in Figure 4. The rectangles represent record types, the ovals - set types. All ovals are white, as there are no regular set types. The lines join record types in a set type. A small rectangle attached to one of the record type rectangles of a set type indicates its owner. A record type of the sole set type is both an owner and a member of that set type and, accordingly, two lines are drawn between the record type rectangle and the set type oval.

Simple lines are used to indicate the singular set type. The multiple univocal set types are pictured by means of a semicircle terminating a line (or lines) connecting a member (or members) of those set types. Finally, a thin line means the partial record type in a set type, and - conversely - a thick line specifies the total record type of the set type.

Figure 4. Diagrammatic Notation for a Logical Network Structure.

For the sake of consistency, we introduced a graphical representation for logical relational structure (Figure 5). But this approach creates problems in and of itself. In the relational model, the single notion - that of table definition - serves both to express entity and relationship kinds of a conceptual structure. However, as pointed out in Section 2, the 'relationship' table definitions must exhibit foreign-to-primary-key matches. Since knowledge of entity and relationship attributes was not explicit in the graphical notation of a conceptual structure, the conversion mechanisms into relational model must access the detailed definitions of entity and relationship kinds. We also conform to an assumption of the standard relational model that the user of a system is knowledgeable of attributes drawn from a common domain, of an attribute role in a table, of existence of functional dependencies between certain attributes.

As shown in Figure 5, a table layout is the single diagrammatic notation for a relational structure. It consists of the name of the table definition and a varying number of columns divided
into three groups: primary key attribute(s), foreign key attribute(s), and the symbols of remaining attributes.

<table>
<thead>
<tr>
<th>B or V</th>
<th>Name of Table Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>PK symbols</td>
</tr>
<tr>
<td>FK</td>
<td>FK symbols</td>
</tr>
<tr>
<td>C</td>
<td>C symbols</td>
</tr>
</tbody>
</table>

B - base table
V - view table
PK - primary key
FK - foreign key
C - remaining attributes

Figure 5 Diagrammatic Notation for a Logical Relational Structure.

4. DERIVATION RULES FROM CONCEPTUAL TO NETWORK DESIGN

The derivation rules recognize that a fully automatic approach to design is not workable. Altogether, the rules constitute a quasi-expert system which proposes feasible transforms while giving a constant scope to designer intervention or even requesting the designer's action. An overall conceptual structure is converted on a step-by-step basis, such that a step is meant to be a particular relationship kind. The CAD-tool keeps track of completed transforms and ensures the coordination among steps and the integration of the logical structure being derived. This approach is easier to implement in that it incrementally builds an overall logical structure; however, it creates the problem of ordering of relationship kinds for integration. Though we admit that the ordering can influence the final solution, the CAD system does not support the designer in this matter. In the future, however, we intend to extend the methodology by dividing relationship kinds into classes based on some integration policy and available statistics (in particular, relative design ranks of functions as defined in Maciaszek (1986)).

The derivation rules are defined separately for logical design of network (this Section) and relational (next Section) databases. They handle the extended range of conceptual constructs, as classified in Section 2. The only constructs that are not subjected to automatic derivation are the nested relationship kinds and the relationship kinds of degree larger than three. Those relationship kinds are uncommon in practice and - if required - can be easily converted to simpler constructs prior to derivation process.

We now present the derivation rules for a range of possible basic conceptual structures. Ten realistic structures have been chosen and transformed into logical network structures (Figures 6 through 15) and into logical relational structures (Section 5 - Figures 16 through 25). The examples of Figures 6 through 25 account for the psychological aspect of pragmatics of the derivation process. Technically, it is realized by clicking the mouse that points to a relationship kind to be converted (after first choosing "Convert Relationship" from the "Choices" menu). The syntactics and semantics of the derivation are explained separately - with reference to an example at hand.

The syntactics function \( \text{syn} = (\text{syn}(\text{con}) \Rightarrow \text{syn}(\text{net})) \) is an implication function, where

\[
\text{syn}(\text{net}) = (\text{syn}(\text{auto}) \cup \text{syn}(\text{user})).
\]

It assigns the names and characteristics of entity and relationship kinds to the names and characteristics of record and set types. The syntactics of network structure that is derived automatically is denoted \( \text{syn}(\text{auto}) \). In some transforms the user is interrogated by the system and complements the derivation syntactics by supplying his/her own
names and characteristics. We denote this activity - syn\(\text{(user)}\).

The semantics function sem = (sem\(\text{(con)}\) ⇒ sem\(\text{(net)}\)) is an equivalence function, where sem\(\text{(net)}\) = (sem\(\text{(auto)}\) ∪ sem\(\text{(user)}\)). It assigns the attributes of entity and relationship kinds to the data-items of record types. As before, sem\(\text{(auto)}\) denotes the portion of semantics that is derived automatically, and sem\(\text{(user)}\) is the remainder arbitrarily defined by the user.

**Figure 6. Transform N1 - Sequential Relationship Kind.**

**N1 - Syntactics:**

\[
\text{syn: } \{\text{SYSTEM} (\text{net}), \text{EMPLOYEE} (\text{net}), \text{FILE} (\text{net})\} : (\text{FILE} (\text{con}) ⇒ \text{FILE} (\text{net})) \land \\
(\text{EMPLOYEE} (\text{con}) ⇒ \text{EMPLOYEE} (\text{net})) \land \text{SYSTEM} (\text{net}) \land (\text{SYSTEM} (\text{net}) \equiv \text{owner}) \land \\
(\text{EMPLOYEE} (\text{net}) \equiv \text{total})\} \cup \{\text{syn} (\text{user}) = \emptyset\}
\]

**N1 - Semantics:**

\[
\text{sem: } \{\text{EMPLOYEE} (\text{net}).\{D\} : \text{EMPLOYEE} (\text{con}).\{A\} \equiv \text{EMPLOYEE} (\text{net}).\{D\}\} \cup \\
\{\text{SYSTEM} (\text{net}).\{D\} : \text{SYSTEM} (\text{net}).\{D\} = (\emptyset \lor \text{any-item})\}
\]

*Note-1:* FILE\(\text{(con)}\) is weak (it does not have attributes).

*Note-2:* Some DBMS-s require that any record type defined in the schema has to contain at least one user data-item ("any-item"). However, this is only applied to the DBMS-s that do not provide for the record type SYSTEM - and force the user to define such a record type in the schema and made the user responsible for its maintenance (e.g. DMS-1100).

**N2 - Syntactics:**

\[
\text{syn: } \{\text{COURSE} (\text{net}), \text{PREREQUISI} (\text{net})\} : (\text{PREREQUISI} (\text{con}) ⇒ \text{PREREQUISI} (\text{net})) \land \\
(\text{COURSE} (\text{con}) ⇒ \text{COURSE} (\text{net})) \land (\text{COURSE} (\text{net}) \equiv \text{owner} \& \text{member}) \land \\
(\text{COURSE} (\text{net}) \equiv \text{partial})\} \cup \{\text{syn} (\text{user}) = \emptyset\}
\]

**N2 - Semantics:**

\[
\text{sem: } \{\text{COURSE} (\text{net}).\{D\} : \text{COURSE} (\text{con}).\{A\} \equiv \text{COURSE} (\text{net}).\{D\}\} \cup
\]
[[\text{COURSE}(\text{net}).\{D\} : | \text{COURSE}(\text{net}).D_u | ]

\textit{Note-1:} Transform N2 works according to a specification rule - first introduced in 1978 - that a single record type may be both owner and member in the same set type. However, currently only a few DBMS-s support this rule (e.g. IDMS).

\textit{Note-2:} Performance considerations may justify adding to \text{COURSE}(\text{net}).\{D\} a data-item \(D_u\), which stands for an identifier of a superior record occurrence in the hierarchy (e.g. \text{SUPERIOR-COURSE-ID}). The notation \(\ldots\) indicates that the action between the two vertical bars is optional and taken arbitrarily by the user.

\textit{Note-3:} An unrealistic assumption was made that any given course can be requested as a prerequisite by one superior course only.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure7.png}
\caption{Transform N2 - Sole Partial Weak Relationship Kind in the Hierarchical Recursion.}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure8.png}
\caption{Transform N3 - Sole Total Regular Relationship Kind in the Homogeneous Network Recursion.}
\end{figure}
N3 - Syntactics:
\[ \{\text{PART}(\text{net}), \text{LINK1}(\text{net}), \text{ASSEMBLY}(\text{net}), \text{COMPONENT}(\text{net})\} : (\text{STRUCTURE}(\text{con}) \Rightarrow \text{LINKi}(\text{net}), i = \text{next natural number}) \land (\text{PART}(\text{con}) \Rightarrow \text{PART}(\text{net})) \land (\text{PART}(\text{net}) = \text{owner in ASSEMBLY}(\text{net})) \land (\text{PART}(\text{net}) = \text{owner in COMPONENT}(\text{net})) \land (\text{LINK1}(\text{net}) = \text{total in ASSEMBLY}(\text{net}) \land COMPONENT(\text{net}) ) \land (\text{PART}(\text{net}) = \text{total in ASSEMBLY}(\text{net}) \land COMPONENT(\text{net})) \land (\text{LINK1}(\text{net}) = \text{total in ASSEMBLY}(\text{net}) \land COMPONENT(\text{net})) \} \cup \{\{\text{ASSEMBLY}(\text{net})\} : (\text{user interaction} \Rightarrow \text{ASSEMBLY}(\text{net})) \land (\{\text{COMPONENT}(\text{net})\} : \text{user interaction} \Rightarrow \text{COMPONENT}(\text{net}))\]  

N3 - Semantics:
\[ \{\text{PART}(\text{net}).\{\text{D}\}, \text{LINK1}(\text{net}).\{\text{D}\}\} : (\text{PART}(\text{con}).\{\text{A}\} = \text{PART}(\text{net}).\{\text{D}\}) \land (\text{STRUCTURE}(\text{con}).\{\text{A}\} = \text{LINK1}(\text{net}).\{\text{D}\}) \} \cup [\text{sem}(\text{user}) = \emptyset] \]

**Figure 9. Transform N4 - Sole Partial Regular Relationship Kind in the Heterogeneous Network Recursion.**

N4 - Syntactics:
\[ \{(\text{PERSON}(\text{net}), \text{TELLER}(\text{net}), \text{CUSTOMER}(\text{net}), \text{LINK2}(\text{net}), \text{GENERIC1}(\text{net}), \text{TEL-L2}(\text{net}), \text{CUS-L2}(\text{net})\} : (\text{PERSON}(\text{con}) \Rightarrow \text{PERSON}(\text{net})) \land (\text{TRANSACTIO}(\text{con}) \Rightarrow \text{LINK2}(\text{net})) \land \text{GENERIC1}(\text{net}) \land \text{TEL-L2}(\text{net}) \land \text{CUS-L2}(\text{net}) \land (\text{PERSON}(\text{net}) = \text{owner in GENERIC1}) \land (\text{TELLER}(\text{net}) = \text{owner in TEL-L2}) \land (\text{CUSTOMER}(\text{net}) = \text{owner in CUS-L2}) \land (\text{TELLER}(\text{net}) = \text{partial in GENERIC1}) \land (\text{CUSTOMER}(\text{net}) = \text{partial in GENERIC1}) \land (\text{LINK2}(\text{net}) = \text{total in TEL-L2}) \land (\text{LINK2}(\text{net}) = \text{total in CUS-L2})\} \cup \{\{\text{TELLER}(\text{net}),} \]
CUSTOMER(net) : (PERSON(con) ⇒ user interaction ⇒ TELLER(net)) ∧
(PERSON(con) ⇒ user interaction ⇒ CUSTOMER(net))]

N4 - Semantics:
sem: [[PERSON(net).{D}, TELLER(net).{D}, CUSTOMER(net).{D}, LINK2(net).{D}]
 : (PERSON(con).{A} ≡ (PERSON(net).{D} ∪ TELLER(net).{D} ∪ CUSTOMER(net).{D}))) ∧ (TRANSACTION(con).{A} ≡ LINK2(net).{D})] ∪
[[LINK2(net).{D}] : (is GENERIC1 of advantage? ⇒ user interaction) ∧
([LINK2(net).{D_t, D_c}])]

Note-1: Transform N4 utilizes the generalization hierarchy. Hence, it introduces the generic set type between CAN-BE record type (PERSON) and IS-A record types (TELLER and CUSTOMER) (v. Maciaszek(1986) for details of our treatment of generalization).

Note-2: Performance considerations may justify enriching LINK2(net).{D} by two data-items D_t and D_c being identifiers of TELLER(net) and CUSTOMER(net), respectively.

Figure 10. Transform N5 - Binary Singular Partial Weak Relationship Kind.

N5 - Syntactics:
syn: [{PROFESSOR(net), OFFICE(net), ALLOCATION(net)} : (ALLOCATION(con) ⇒ ALLOCATION(net)) ∧ (PROFESSOR(con) ⇒ PROFESSOR(net)) ∧
(OFFICE(con) ⇒ OFFICE(net)) ∧ (PROFESSOR(net) ≡ partial) ∧ (OFFICE(net) ≡ partial)] ∪ [{PROFESSOR(net)} : user interaction ⇒ PROFESSOR(net) = owner]

N5 - Semantics:
sem: [{PROFESSOR(net).{D}, OFFICE(net).{D}} : (PROFESSOR(con).{A} =
PROFESSOR(net).{D}) \land (OFFICE(con).{A} \equiv OFFICE(net).{D})] \cup [sem(user) = \emptyset]

Figure 11. Transform N6 - Binary Univocal Total Weak Relationship Kind.

N6 - Syntactics:
\[\text{syn: } \{\{\text{DEPARTMENT(net), EMPLOYEE(net), EMPLOYMENT(net)} \} : \]
\[(\text{EMPLOYMENT(con) } \Rightarrow \text{EMPLOYMENT(net)}) \land (\text{DEPARTMENT(con) } \Rightarrow \text{DEPARTMENT(net)}) \land (\text{EMPLOYEE(con) } \Rightarrow \text{EMPLOYEE(net)}) \land (\text{DEPARTMENT(net) } \equiv \text{owner}) \land (\text{DEPARTMENT(net) } \equiv \text{total}) \land (\text{EMPLOYEE(net) } \equiv \text{total})] \cup [\text{syn(user) } = \emptyset]\]

N6 - Semantics:
\[\text{sem: } \{\{\text{DEPARTMENT(net).{D}, EMPLOYEE(net).{D}} : (\text{DEPARTMENT(con).{A} } \equiv \text{DEPARTMENT(net).{D}}) \land (\text{EMPLOYEE(con).{A} } \equiv \text{EMPLOYEE(net).{D}})] \cup [\text{sem(user) } = \emptyset]\]

N7 - Syntactics:
\[\text{syn: } \{\{\text{STUDENT(net), COURSE(net), LINK3(net), STU-L3(net), COU-L3(net)} \} : \]
\[(\text{ENROLLMENT(con) } \Rightarrow \text{LINK3(net)}) \land (\text{STUDENT(con) } \Rightarrow \text{STUDENT(net)}) \land (\text{COURSE(con) } \Rightarrow \text{COURSE(net)}) \land \text{STU-L3(net) } \land \text{COU-L3(net) } \land (\text{STUDENT(net) } \equiv \text{owner in STU-L3}) \land (\text{COURSE(net) } \equiv \text{owner in COU-L3}) \land (\text{STUDENT(net) } \equiv \text{partial}) \land (\text{COURSE(net) } \equiv \text{partial}) \land (\text{LINK3(net) } \equiv \text{total in STU-L3}) \land (\text{LINK3(net) } \equiv \text{total in COU-L3})] \cup [\text{syn(user) } = \emptyset]\]
Figure 12. Transform N7 - Binary Nonunivocal Partial Regular Relationship Kind.

N7 - Semantics:
\[
\text{sem: } \left\{ \text{STUDENT}(\text{net}).\{D\}, \text{COURSE}(\text{net}).\{D\}, \text{LINK3}(\text{net}).\{D\} \right\} \cup \\
\left( \text{ENROLLMENT}(\text{con}).\{A\} \equiv \text{LINK3}(\text{net}).\{D\} \right) \land \\
\left( \text{STUDENT}(\text{con}).\{A\} \equiv \text{STUDENT}(\text{net}).\{D\} \right) \land \\
\left( \text{COURSE}(\text{con}).\{A\} \equiv \text{COURSE}(\text{net}).\{D\} \right) \cup \\
\left\{ \text{LINK3}(\text{net}).\{D\} : \text{LINK3}(\text{net}).\{D_s, D_c\} \right\}
\]

Note-1: See Note-2 on Transform N4

Figure 13. Transform N8 - Ternary Singular Partial Regular Relationship Kind.
N8 - Syntactics:
\[
\text{syn: } \{\{\text{RESIDENCE(net), MAN(net), WOMAN(net), MARRIAGE(net)} \} : (MARRIAGE(con) \Rightarrow MARRIAGE(net)) \land (RESIDENCE(con) \Rightarrow RESIDENCE(net)) \land (MAN(con) \Rightarrow MAN(net)) \land (WOMAN(con) \Rightarrow WOMAN(net)) \land (RESIDENCE(net) \equiv \text{partial}) \land (MAN(net) \equiv \text{partial}) \land (WOMAN(net) \equiv \text{partial}) \} \cup \{\{\text{RESIDENCE(net)} \} : \text{user interaction} \Rightarrow (RESIDENCE(net) \equiv \text{owner})\}
\]

N8 - Semantics:
\[
\text{sem: } \{\{\text{RESIDENCE(net).}\{D\}, MAN(net).\{D\}, WOMAN(net).\{D\}) : (RESIDENCE(con).\{A\} \equiv RESIDENCE(net).\{D\}) \land (MAN(con).\{A\} \equiv MAN(net).\{D\}) \land (WOMAN(con).\{A\} \equiv WOMAN(net).\{D\}) \} \cup \{\{\text{RESIDENCE(net).}\{D\}, MAN(net).\{D\}, WOMAN(net).\{D\}) : \text{user interaction} \Rightarrow (MARRIAGE(con).\{A\} \equiv \{D\}^r : \{(D\}^r \cup \text{RESIDENCE(net).}\{D\}) \lor \{(D\}^r \cup \text{MAN(net).}\{D\}) \lor \{(D\}^r \cup \text{WOMAN(net).}\{D\})\}\}
\]

\textbf{Note-1: } \{D\}^r \text{ is the set of data-items equivalent to the set of attributes MARRIAGE(con).}\{A\}.

---

Figure 14. Transform N9 - Ternary Univocal Partial Regular Relationship Kind.

N9 - Syntactics:
\[
\text{syn: } \{\{\text{SUPERVISOR(net), EMPLOYEE(net), PROJECT(net), JOB(net)} : (JOB(con) \Rightarrow JOB(net)) \land (SUPERVISOR(con) \Rightarrow SUPERVISOR(net)) \land (EMPLOYEE(con) \Rightarrow EMPLOYEE(net)) \land (PROJECT(con) \Rightarrow PROJECT(net)) \land (SUPERVISOR(net) \equiv \text{owner}) \land (SUPERVISOR(net) \equiv \text{partial}) \land
\]

---
(EMPLOYEE(net) \equiv \text{partial}) \land (PROJECT(net) \equiv \text{partial}) \cup \{\text{syn(user)} = \emptyset\}

N9 - Semantics:
sem: [SUPERVISOR(net).{D}, EMPLOYEE(net).{D}, PROJECT(net).{D}] : 
((SUPERVISOR(con).{A} \cup JOB(con).{A}) \equiv SUPERVISOR(net).{D}) \land 
(EMPLOYEE(con).{A} \equiv EMPLOYEE(net).{D}) \land (PROJECT(con).{A} \equiv PROJECT(net).{D}) \cup \{\text{sem(user)} = \emptyset\}

Figure 15. Transform N10 - Ternary Nonunivocal Partial Regular Relationship Kind.

N10 (Choice 1) - Syntactics:
syn: \{\{\text{MANUFACTUR(net)}, \text{CUSTOMER(net)}, \text{MIDDLEMAN(net)}, \text{LINK4(net)}, \text{MAN-L4(net)}, \text{CUS-L4(net)}, \text{MID-L4(net)}\} : \{\text{LEASING(con)} \Rightarrow \text{LINK4(net)}\} \land 
(\text{MANUFACTUR(con)} \Rightarrow \text{MANUFACTUR(net)}) \land (\text{CUSTOMER(con)} \Rightarrow \text{CUSTOMER(net)}) \land (\text{MIDDLEMAN(con)} \Rightarrow \text{MIDDLEMAN(net)}) \land
(MANUFACTUR(net) == owner in MAN-L4) ∧ (CUSTOMER(net) == owner in CUS-L4) ∧ (MIDDLEMAN(net) == owner in MID-L4) ∧ (MANUFACTUR(net) == partial in MAN-L4) ∧ (CUSTOMER(net) == partial in CUS-L4) ∧ (MIDDLEMAN(net) == partial in MID-L4) ∧ (LINK4(net) == total in MAN-L4 & CUS-L4 & MID-L4) \) \cup \ [(\text{syn(user)} = \emptyset)]

N10 (Choice 1) - Semantics:
\[
\text{sem: } \{\{\text{MANUFACTUR(net).} \{D\}, \text{CUSTOMER(net).} \{D\}, \text{MIDDLEMAN(net).} \{D\}, \text{LINK4(net).} \{D\}\} : (\text{LEASING(con).} \{A\} \equiv \text{LINK4(net).} \{D\}) \land \\
(\text{MANUFACTUR(con).} \{A\} \equiv \text{MANUFACTUR(net).} \{D\}) \land \\
(\text{CUSTOMER(con).} \{A\} \equiv \text{CUSTOMER(net).} \{D\}) \land (\text{MIDDLEMAN(con).} \{A\} \equiv \\
\text{MIDDLEMAN(net).} \{D\}) \} \cup \{\{\text{LINK4(net).} \{D\}\} : \text{LINK4(net).} \{D\_ma, D\_c, D\_mi\}\}
\]

\text{Note-1: Performance considerations may justify enriching } \text{LINK4(net).} \{D\} \text{by three data-items } D\_ma, D\_c, \text{and } D\_mi, \text{that is the identifiers of } \text{MANUFACTUR(net), CUSTOMER(net), and MIDDLEMAN(net).}

N10 (Choice 2) - Syntactics:
\[
\text{syn: } \{\{\text{MANUFACTUR(net).} \{D\}, \text{CUSTOMER(net).} \{D\}, \text{MIDDLEMAN(net).} \{D\}, \text{LINK4(net), LINK5(net), LINK6(net), MAN-L4(net), CUS-L4(net), MID-L5(net), MAN-L5(net), CUS-L6(net), MID-L6(net)}\} : (\text{LEASING(con) } \Rightarrow (\text{LINK4(net) } \cup \\
\text{LINK5(net) } \cup \text{LINK6(net)})) \land (\text{MANUFACTUR(con) } \Rightarrow \text{MANUFACTUR(net)}) \land \\
(\text{CUSTOMER(con) } \Rightarrow \text{CUSTOMER(net)}) \land (\text{MIDDLEMAN(con) } \Rightarrow \\
\text{MIDDLEMAN(net)}) \land (\text{MANUFACTUR(net) } \equiv \text{owner in MAN-L4 } \land \text{MAN-L5}) \land \\
(\text{CUSTOMER(net) } \equiv \text{owner in CUS-L4 } \land \text{CUS-L6}) \land (\text{MIDDLEMAN(net) } \equiv \text{owner in } \text{MID-L5 } \land \text{MID-L6}) \land (\text{MANUFACTUR(net) } \equiv \text{partial in MAN-L4 } \land \text{MAN-L5}) \land \\
(\text{CUSTOMER(net) } \equiv \text{partial in CUS-L4 } \land \text{CUS-L6}) \land (\text{MIDDLEMAN(net) } \equiv \text{partial in MID-L5 } \land \text{MID-L6}) \land (\text{LINK4(net) } \equiv \text{total in MAN-L4 } \land \text{CUS-L4}) \land \\
(\text{LINK5(net) } \equiv \text{total in MAN-L5 } \land \text{MID-L5}) \land (\text{LINK6(net) } \equiv \text{total in CUS-L6 } \land \text{MID-L6}) \} \cup \ [(\text{syn(user)} = \emptyset) \lor (\text{syn(user)} \equiv \text{user})] \]

N10 (Choice 2) - Semantics:
\[
\text{sem: } \{\{\text{MANUFACTUR(net).} \{D\}, \text{CUSTOMER(net).} \{D\}, \text{MIDDLEMAN(net).} \{D\}, \text{LINK4(net), LINK5(net), LINK6(net), MAN-L4(net), CUS-L4(net), MID-L5(net), MAN-L5(net), CUS-L6(net), MID-L6(net)}\} : (\text{MANUFACTUR(con).} \{A\} \equiv \\
\text{MANUFACTUR(net).} \{D\}) \land (\text{CUSTOMER(con).} \{A\} \equiv \text{CUSTOMER(net).} \{D\}) \land \\
(\text{MIDDLEMAN(con).} \{A\} \equiv \text{MIDDLEMAN(net).} \{D\}) \} \cup \ [(\text{user interaction } \Rightarrow \\
(\text{LEASING(con).} \{A\} \equiv \text{LINK4(net).} \{D\} \cup \text{LINK5(net).} \{D\} \cup \text{LINK6(net).} \{D\})) \land \\
(\{\text{LINK4(net).} \{D\}, \text{LINK5(net).} \{D\}, \text{LINK6(net).} \{D\}\} : \text{is asymmetric structure workable? } \Rightarrow \text{user interaction } \Rightarrow (\text{delete LINK4(net) } \lor \text{delete LINK5(net) } \lor \text{delete LINK6(net)})] \]

\text{Note-1: If asymmetric structure is semantically equivalent to the symmetric one as shown in Figure 15, then the feedback to the syntactics rule will simplify the structure by eliminating either LINK4(net) or LINK5(net) or LINK6(net), together with the pertinent set types.}
5. DERIVATION RULES FROM CONCEPTUAL TO RELATIONAL DESIGN

Before we proceed with the derivation rules for the relational model, a note on a fundamental difference between the network and relational approach to the logical design is necessary. The logical schema design in the relational environment is by order of magnitude less crucial than in the network setting. The fact that the relational data definition statements constitute a part of the common dual-mode database language (i.e. definition and manipulation) and as such can be executed at any time, makes it not necessary to go through the total design process before starting the system up. That is to say - at least theoretically (cp. Codd(1986)). In practice, the low fidelity of the relational DBMS-s to the model entails quite significant restrictions on the system's flexibility and its dynamic modifications. As a result of the current state of art of the relational technology, the latest proposed ANSI standard has gone as far as to take a static approach to relational database description and it has specified two separate interfaces to a DBMS: (1) schema definition language, and (2) module language and data manipulation language (Draft, 1985b).

In these circumstances we have to state clearly the relevance of our derivation rules to the level of sophistication of relational technology. In short, the design principles that we enforce surpass the current DBMS-s (including DB2 - v. Date (1986)) in a number of respects but do not reach the full fidelity of relational model. We therefore assume that the domains and the primary and foreign keys of relations are well understood, that the concept of referential integrity is enforced, and the nulls in a limited sense are supported. We do not expect the system to fully conform to the view updating rule but we assume that the derived view tables are updatable. While this last assumption is quite strong, it is softened by the fact that at the design stage only weak relationship kinds (with the exception of sole relationship kinds) can be subjected to conversion to view tables (regular relationship kinds constitute base tables).

Methodologically speaking there is another important difference between the network and relational approach to logical design. The difference rises from the information rule of the relational model. The rule states that all information in a database is represented explicitly at the logical level and in exactly one way - by values in tables. (Incidentally, this has a cumbersome effect put aptly down by Hartzband and Maryanski (1985): "In the relational model, tables holding the keys of the related objects can be used to achieve a kind of explicit relationship support, similar to using more elaborate nouns in a verb-free language - the system is functional although the structure is somewhat obtuse.") As a result, we are unable to assume (as we have done for the network derivation) the system's only implicit knowledge of attributes - this knowledge has to be explicit in the system and to manifest itself in the definition of relation columns.

The derivation rules for the relational model are shown in Figures 16 through 25 and are based on the same examples as used for the network logical transforms. (Note that at this stage the derivation rules cope with the normalized conceptual objects only. The denormalized objects (if any - v. Maciaszek (1986)) have to be converted manually and fed to the system.)

R1 - Syntactics:

syn: [ {EMPLOYEE(rel)} : ((FILE(con) \cup EMPLOYEE(con)) \Rightarrow EMPLOYEE(rel)) \land (EMPLOYEE(rel) \equiv base) ] \cup [syn(user) = \emptyset]

R1 - Semantics:

sem: [ {EMPLOYEE(rel).{C}} : (EMPLOYEE(con).{A} \equiv EMPLOYEE(rel).{C}) \land (EMPLOYEE(con).PK \equiv EMPLOYEE(rel).PK) ] \cup [sem(user) = \emptyset]
Figure 16. Transform R1 - Sequential Relationship Kind.

<table>
<thead>
<tr>
<th>B</th>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>C</td>
</tr>
<tr>
<td>EMPSYM</td>
<td></td>
</tr>
</tbody>
</table>

Figure 17. Transform R2 - Sole Partial Weak Relationship Kind in the Hierarchical Recursion.

R2 (Choice 1) - Syntactics:

\[ \text{syn: } \{ \{\text{COURSE}(\mathrm{rel}), \text{PREREQUISI}(\mathrm{rel})\} : (\text{COURSE}(\mathrm{con}) \Rightarrow \text{COURSE}(\mathrm{rel})) \land \\
(\text{PREREQUISI}(\mathrm{con}) \Rightarrow \text{PREREQUISI}(\mathrm{rel})) \land (\text{COURSE}(\mathrm{rel}) \equiv \text{base}) \land \\
(\text{PREREQUISI}(\mathrm{rel}) \equiv \text{base}) \} \cup \{\text{syn}(\text{user}) = \emptyset\} \]

R2 (Choice 1) - Semantics:

\[ \text{sem: } [ \{\text{COURSE}(\mathrm{rel}).\{\mathrm{C}\}, \text{PREREQUISI}(\mathrm{rel}).\{\mathrm{C}\}\} : (\text{COURSE}(\mathrm{con}).\{\mathrm{A}\} \equiv \\
\]
COURSE(rel).{C}) \land (COURSE(con).PK \equiv COURSE(rel).PK \equiv PREREQUISI(rel).FK) \cup \{(PREREQUISI(rel).{C}) : PREREQUISI(rel).PK? \Rightarrow user interaction \Rightarrow (PK symbol : PK & FK drawn from common domain, \forall n [\{PK \neq FK\} \lor (FK = null)]\}

R2 (Choice 2) - Syntactics:

\text{syn}: \{(COURSE(rel)) : (COURSE(con) \land PREREQUISI(con)) \Rightarrow COURSE(rel) \land (COURSE(rel) \equiv base) \cup \{\text{syn(user) = } \emptyset\}\}

R2 (Choice 2) - Semantics:

\text{sem}: \{(COURSE(rel).{C}) : (COURSE(con).{A} \equiv COURSE(rel).{C}) \land (COURSE(con).PK \equiv COURSE(rel).FK) \cup \{(COURSE(rel).{C}) : COURSE(rel).PK? \Rightarrow user interaction \Rightarrow (PK symbol : PK & FK drawn from common domain, \forall n [\{PK \neq FK\} \lor (FK = null)]\}

Figure 18. Transform R3 - Sole Total Regular Relationship Kind in the Homogeneous Network Recursion.

R3 - Syntactics:

\text{syn}: \{(PART(rel), STRUCTURE(rel)) : (PART(con) \Rightarrow PART(rel).{C}) \land (STRUCTURE(con) \Rightarrow STRUCTURE(rel)) \land (PART(rel) \equiv base) \land (STRUCTURE(rel) \equiv base) \cup \{\text{syn(user) = } \emptyset\}\}

R3 - Semantics:

\text{sem}: \{\{PART(rel).{C}, STRUCTURE(rel).{C}) : (PART(con).{A} \equiv PART(rel).{C}) \land (STRUCTURE(con).{A} \equiv STRUCTURE(rel).{C}) \land (PART(con).PK \equiv
PART(rel).PK \land (\text{STRUCTURE}(rel).PK = \text{compound}) \cup 
\{(\text{STRUCTURE}(rel).\{C\} : \text{STRUCTURE}(rel).PK? \Rightarrow \text{user interaction} \Rightarrow (PK = \{FK_1, FK_2\} : FK_1, FK_2, \text{PART}(rel).PK \text{ drawn from common domain, } \forall n [FK_1 \neq FK_2])\}

\text{Note 1: This is one of the transforms that the relational DBMS-s in their current form do not handle well (Date(1986) gives an exercise that yields an answer proving that the current relational support for recursive structures is sadly ad hoc and inefficient).}

![Figure 19. Transform R4 - Sole Partial Regular Relationship Kind in the Heterogeneous Network Recursion.](image)

\textbf{R4 - Syntactics:}
\text{syn: } \{ \text{PERSON(rel), TRANSACTIO(rel), ROLE(rel)} : (\text{PERSON(con) }\Rightarrow \text{PERSON(rel)}) \land (\text{TRANSACTIO(con) }\Rightarrow \text{TRANSACTIO(rel)}) \land (\text{PERSON(rel) }\equiv \text{base}) \land (\text{ROLE(rel) }\equiv \text{base}) \land (\text{TRANSACTIO(rel) }\equiv \text{base}) \} \cup \{\{\text{ROLE(rel)} : (\text{PERSON(con) }\land \text{TRANSACTIO(con)}) \Rightarrow \text{user interaction} \Rightarrow \text{ROLE(rel)}\}

\textbf{R4 - Semantics:}
\text{sem: } \{ \text{PERSON(rel).}\{C\}, \text{TRANSACTIO(rel).}\{C\}, \text{ROLE(rel).}\{C\} : (\text{PERSON(con).}\{A\} \equiv (\text{PERSON(rel).}\{C\} \cup \text{ROLE(rel).}\{C\})) \land (\text{TRANSACTIO(con).}\{A\} \equiv \text{TRANSACTIO(rel).}\{C\}) \land (\text{PERSON(con).PK} \equiv \text{PERSON(rel).PK} \equiv \text{ROLE(rel).FK} : (\text{PK} \supset \text{FK} \land \text{TRANSACTIO(con).PK} \equiv \text{TRANSACTIO(rel).PK})) \} \cup \{\{\text{TRANSACTIO(rel).}\{C\}, \text{ROLE(rel).}\{C\} :
(ROLE(rel).PK \Leftrightarrow \text{user interaction}) \land (\text{TRANSACTIO(rel).}\{FK_1, FK_2\} \Leftrightarrow \text{user interaction}) \land (\text{PERSON(rel).PK, ROLE(rel).FK, TRANSACTIO(rel).}\{FK_1, FK_2\}
drawn from common domain) \land (\exists n [\text{TRANSACTIO(rel).FK}_1 = 
\text{TRANSACTIO(rel).FK}_2 ]) \land (\forall n [\text{TRANSACTIO(rel).FK}_1 \neq \text{null,}
\text{TRANSACTIO(rel).FK}_2 \neq \text{null}])

\text{Note 1: Transform R4 - as any transform involving generalization hierarchy - can have alternative resolutions. We claim, however, that the one presented is most universal and applicable for all heterogeneous network recursions of which we have been able to conceive (cp. Maciaszek (1986)).}

---

\textbf{Choices}

<table>
<thead>
<tr>
<th>Choices</th>
<th>Choice 1:</th>
<th>Choice 2:</th>
<th>Choice 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFESSOR</td>
<td>PROFESSOR PK C PRONAM</td>
<td>OFFICE PK C OFFNUM</td>
<td>ALLOCATION PK FK PRONAM OFFNUM</td>
</tr>
<tr>
<td>ALLOCATION</td>
<td>OFFICE PK C OFFNUM</td>
<td>ALLOCATION PK FK PRONAM OFFNUM</td>
<td></td>
</tr>
<tr>
<td>OFFICE</td>
<td>OFFICE PK C OFFNUM</td>
<td>ALLOCATION PK FK PRONAM OFFNUM</td>
<td></td>
</tr>
</tbody>
</table>

\textbf{Figure 20. Transform R5 - Binary Singular Partial Weak Relationship Kind.}

R5 (Choice 1) - Syntactics:

\text{syn:} [ \{\text{PROFESSOR(rel), OFFICE(rel), ALLOCATION(rel)}\} : (\text{PROFESSOR(con) } \Rightarrow 
\text{PROFESSOR(rel)} ) \land (\text{OFFICE(con) } \Rightarrow \text{OFFICE(rel)} ) \land (\text{ALLOCATION(con) } \Rightarrow
ALLOCATION(rel) ∧ (PROFESSOR(rel) ≡ base) ∧ (OFFICE(rel) ≡ base) ∧
(ALLOCATION(rel) ≡ base) ∪ [syn(user) = ∅]

R5 (Choice 1) - Semantics:
sem: [{PROFESSOR(rel), OFFICE(rel), ALLOCATION(rel)} : (PROFESSOR(con).A ⊆ PROFESSOR(rel).A) ∧ (OFFICE(con).A ⊆ OFFICE(rel).A) ∧
(ADDRESS(con).A ⊆ ADDRESS(rel).A)] ∪ [syn(user) = 0]

R5 (Choice 2) - Syntactics:
syn: [ {PROFESSOR(rel), OFFICE(rel), ALLOCATION(rel)} : ((PROFESSOR(con) ∪
ALLOCATION(con)) ⇔ PROFESSOR(rel)) ∧ (OFFICE(con) ⇔ OFFICE(rel)) ∧
(ALLOCATION(con) ⇔ ALLOCATION(rel)) ∧ (PROFESSOR(rel) ≡ base) ∧
(OFFICE(rel) ≡ base) ∧ (ALLOCATION(rel) ≡ view)] ∪ [syn(user) = (∅ ∨ feedback
from sem(user))]

R5 (Choice 2) - Semantics:
(OFFICE(con).A ⊆ OFFICE(rel).A) ∧
(ALLOCATION(con).A ⊆ ADDRESS(rel).A)] ∪ [syn(user) = 0 ∨ feedback
from sem(user)]

R5 (Choice 3) - Syntactics:
syn: [ {PROFESSOR(rel), OFFICE(rel), ALLOCATION(rel)} : ((OFFICE(con) ∪
ALLOCATION(con)) ⇒ OFFICE(rel)) ∧ (PROFESSOR(con) ⇒ PROFESSOR(rel)) ∧
(ALLOCATION(con) ⇒ ALLOCATION(rel)) ∧ (PROFESSOR(rel) ≡ base) ∧
(OFFICE(rel) ≡ base) ∧ (ALLOCATION(rel) ≡ view)] ∪ [syn(user) = (∅ ∨ feedback
from sem(user))]

R5 (Choice 3) - Semantics:
(OFFICE(con).A ⊆ OFFICE(rel).A) ∧
(ALLOCATION(con).A ⊆ ADDRESS(rel).A)] ∪ [syn(user) = 0 ∨ feedback
from sem(user)]
PROFESSOR(rel).PK = (ALLOCATION(rel).PK ∨ ALLOCATION(rel).FK)) ∧ (OFFICE(con).PK = OFFICE(rel).PK = (ALLOCATION(rel).PK ∨ ALLOCATION(rel).FK)) ∪ [[ALLOCATION(rel).{C}: is view ALLOCATION desirable? ⇒ (create view table ∨ drop view table and feedback to syn(user)]

Figure 21. Transform R6 - Binary Univocal Total Weak Relationship Kind.

R6 (Choice 1) - Syntactics:

\[
\text{syn: } [\{\text{DEPARTMENT}(\text{rel}), \text{EMPLOYEE}(\text{rel}), \text{EMPLOYMENT}(\text{rel})\} : \\
(\text{DEPARTMENT}(\text{con}) \Rightarrow \text{DEPARTMENT}(\text{rel})) ∧ (\text{EMPLOYEE}(\text{con}) \Rightarrow \text{EMPLOYEE}(\text{rel})) ∧ (\text{EMPLOYMENT}(\text{con}) \Rightarrow \text{EMPLOYMENT}(\text{rel})) ∧ \\
(\text{DEPARTMENT}(\text{rel}) \equiv \text{base}) ∧ (\text{EMPLOYEE}(\text{rel}) \equiv \text{base}) ∧ (\text{EMPLOYMENT}(\text{rel}) \equiv \text{base})] ∪ [\text{syn(user)} = \emptyset]
\]

R6 (Choice 1) - Semantics:

\[
\text{sem: } [\{\text{DEPARTMENT}(\text{rel}).\{C\}, \text{EMPLOYEE}(\text{rel}).\{C\}, \text{EMPLOYMENT}(\text{rel}).\{C\}\} : \\
(\text{DEPARTMENT}(\text{con}).\{A\} \equiv \text{DEPARTMENT}(\text{rel}).\{C\}) ∧ (\text{EMPLOYEE}(\text{con}).\{A\} \equiv \text{EMPLOYEE}(\text{rel}).\{C\}) ∧ ((\text{DEPARTMENT}(\text{con}).\{PK\} ∪ \text{EMPLOYEE}(\text{con}).\{PK\}) \equiv \text{EMPLOYMENT}(\text{rel}).\{C\}) ∧ (\text{DEPARTMENT}(\text{con}).\{PK\} \equiv \text{DEPARTMENT}(\text{rel}).\{PK\}) ∧ (\text{EMPLOYEE}(\text{con}).\{PK\} \equiv \text{EMPLOYEE}(\text{rel}).\{PK\}) ∧ (\text{EMPLOYMENT}(\text{rel}).\{PK\} \equiv \text{EMPLOYMENT}(\text{rel}).\{PK\}) →
\]
EMPLOYMENT(rel).FK \rightarrow \text{stands for "functionally determines")} \cup \{\text{sem(user) = } \emptyset\}

R6 (Choice 2) - Syntactics:
\[
\text{syn: } [\{\text{DEPARTMENT}\text{(rel)}, \text{EMPLOYEE}\text{(rel)}, \text{EMPLOYMENT}\text{(rel)}\} :
(\text{DEPARTMENT}\text{(con) }\cup\text{EMPLOYMENT}\text{(con)}) \Rightarrow \text{DEPARTMENT}\text{(rel)}) \land
(\text{EMPLOYEE}\text{(con) }\Rightarrow \text{EMPLOYEE}\text{(rel)} ) \land
(\text{EMPLOYMENT}\text{(con) }\Rightarrow \text{EMPLOYMENT}\text{(rel)} ) \land
(\text{DEPARTMENT}(\text{rel} ) = \text{base}) \land
(\text{EMPLOYEE}(\text{rel} ) = \text{base}) \land
(\text{EMPLOYMENT}(\text{rel} ) = \text{view})] \cup \{\text{syn(user) = } (\emptyset \lor \text{feedback from sem(user)})\}
\]

R6 (Choice 2) - Semantics:
\[
\text{sem: } [\{\text{DEPARTMENT}\text{(rel)}.\{C\}, \text{EMPLOYEE}\text{(rel)}.\{C\}, \text{EMPLOYMENT}\text{(rel)}.\{C\} :
(\text{DEPARTMENT}\text{(con).}\{A\} \cup \text{EMPLOYEE}\text{(con).}\{PK\}) \Rightarrow \text{DEPARTMENT}\text{(rel).}\{C\}) \land
(\text{EMPLOYEE}\text{(con).}\{A\} \Rightarrow \text{EMPLOYEE}\text{(rel).}\{C\}) \land
(\text{DEPARTMENT}\text{(con).}\{PK\} \cup \text{EMPLOYEE}\text{(con).}\{PK\}) \Rightarrow \text{EMPLOYMENT}\text{(rel).}\{C\}) \land
(\text{DEPARTMENT}\text{(con).}\{PK\} = \text{DEPARTMENT}\text{(rel).}\{PK\}) \land
(\text{EMPLOYEE}\text{(con).}\{PK\} = \text{EMPLOYEE}\text{(rel).}\{PK\}) \land
(\text{DEPARTMENT}\text{(rel).}\{PK\} = \text{EMPLOYMENT}\text{(rel).}\{PK\})] \cup [\{\text{EMPLOYMENT}\text{(rel)}.\{C\} : is \text{EMPLOYMENT} \text{ desirable? } \Rightarrow (\text{create view table } \lor \text{drop view table and feedback to } \text{syn(user)})\]
\]

**Choices**

Figure 22. Transform R7 - Binary Nonunivocal Partial Regular Relationship Kind.

R7 - Syntactics:
\[
\text{syn: } [\{\text{STUDENT}\text{(rel)}, \text{COURSE}\text{(rel)}, \text{ENROLLMENT}\text{(rel)}\} : (\text{STUDENT}\text{(con) }\Rightarrow
\text{STUDENT}\text{(rel)}) \land (\text{COURSE}\text{(con) }\Rightarrow \text{COURSE}\text{(rel)}) \land (\text{ENROLLMENT}\text{(con) }\Rightarrow
\text{ENROLLMENT}\text{(rel)}) \land (\text{STUDENT}\text{(rel) }= \text{base}) \land (\text{COURSE}\text{(rel) }= \text{base}) \land
(\text{ENROLLMENT}\text{(rel) }= \text{base})] \cup \{\text{syn(user) = } (\emptyset\}
\]
R7 - Semantics:
\[
\text{sem: } \left\{ \text{STUDENT}(\text{rel}).\{C\}, \text{COURSE}(\text{rel}).\{C\}, \text{ENROLLMENT}(\text{rel}).\{C\} \right\} : \\
(\text{STUDENT}(\text{con}).\{A\} \equiv \text{STUDENT}(\text{rel}).\{C\}) \land (\text{COURSE}(\text{con}).\{A\} \equiv \text{COURSE}(\text{rel}).\{C\}) \land (\text{ENROLLMENT}(\text{con}).\{A\} \equiv \text{ENROLLMENT}(\text{rel}).\{C\}) \land (\text{STUDENT}(\text{con}).\{PK\} \equiv \text{STUDENT}(\text{rel}).\{PK\}) \land (\text{COURSE}(\text{con}).\{PK\} \equiv \text{COURSE}(\text{rel}).\{PK\}) \\
\left(\text{STUDENT}(\text{con}).\{PK\} \equiv \text{STUDENT}(\text{rel}).\{PK\}) \land (\text{COURSE}(\text{con}).\{PK\} \equiv \text{COURSE}(\text{rel}).\{PK\}) \land (\text{ENROLLMENT}(\text{con}).\{PK\} \equiv \text{ENROLLMENT}(\text{rel}).\{PK\}) \right) \cup \\
\left[\text{sem(user)} = \emptyset\right]
\]

R8 - Syntactics:
\[
\text{syn: } \left\{ \text{RESIDENCE}(\text{rel}), \text{MAN}(\text{rel}), \text{WOMAN}(\text{rel}), \text{MARRIAGE}(\text{rel}) \right\} : \\
(\text{RESIDENCE}(\text{con}) \\
\Rightarrow \text{RESIDENCE}(\text{rel}) \land (\text{MAN}(\text{con}) \Rightarrow \text{MAN}(\text{rel})) \land (\text{WOMAN}(\text{con}) \Rightarrow \text{WOMAN}(\text{rel}) \land (\text{MARRIAGE}(\text{con}) \Rightarrow \text{MARRIAGE}(\text{rel}) \land (\text{RESIDENCE}(\text{rel}) \equiv \text{base}) \land (\text{MAN}(\text{rel}) \equiv \text{base}) \land (\text{WOMAN}(\text{rel}) \equiv \text{base}) \land (\text{MARRIAGE}(\text{rel}) \equiv \text{base}) \right) \cup \\
\left[\text{syn(user)} = \emptyset\right]
\]

R8 - Semantics:
\[
\text{sem: } \left\{ \text{RESIDENCE}(\text{rel}).\{C\}, \text{MAN}(\text{rel}).\{C\}, \text{WOMAN}(\text{rel}).\{C\}, \text{MARRIAGE}(\text{rel}).\{C\} \right\} : \\
(\text{RESIDENCE}(\text{con}).\{A\} \equiv \text{RESIDENCE}(\text{rel}).\{C\}) \land (\text{MAN}(\text{con}).\{A\} \equiv \text{MAN}(\text{rel}).\{C\}) \land (\text{WOMAN}(\text{con}).\{A\} \equiv \text{WOMAN}(\text{rel}).\{C\}) \land (\text{RESIDENCE}(\text{con}).\{PK\} \equiv \text{MAN}(\text{con}).\{PK\}) \land (\text{MAN}(\text{con}).\{PK\} \equiv \text{WOMAN}(\text{con}).\{PK\}) \land (\text{WOMAN}(\text{con}).\{PK\} \equiv \emptyset) \\
\]

Figure 24. Transform R9 - Ternary Univocal Partial Regular Relationship Kind.

R9 - Syntactics:
syn: [{SUPERVISOR(reI), EMPLOYEE(reI), PROJECT(reI), JOB(reI)}:
(SUPERVISOR(con) ⇒ SUPERVISOR(reI)) ∧ (EMPLOYEE(con) ⇒
EMPLOYEE(reI)) ∧ (PROJECT(con) ⇒ PROJECT(reI) ∧ (JOB(con) ⇒ JOB(reI)) ∧
(SUPERVISOR(reI) ≡ base) ∧ (EMPLOYEE(reI) ≡ base) ∧ (PROJECT(reI) ≡ base) ∧
(JOB(reI) ≡ base)] ∪ [syn(user) = ∅]

R9 - Semantics:
sem: [{ SUPERVISOR(reI).{C}, EMPLOYEE(reI).{C}, PROJECT(reI).{C},
JOB(reI).{C}): (SUPERVISOR(con).{A} ≡ SUPERVISOR(reI).{C}) ∧
(EMPLOYEE(con).{A} ≡ EMPLOYEE(reI).{C}) ∧ (PROJECT(con).{A} ≡
PROJECT(reI).{C}) ∧ ((SUPERVISOR(con).PK ∪ EMPLOYEE(con).PK ∪
PROJECT(con).PK ∪ JOB(con).{A}) ≡ JOB(reI).{C}) ∧ (SUPERVISOR(con).PK
((EMPLOYEE(reI).PK ∪ PROJECT(reI).PK) ≡ JOB(reI).PK) ∧ (JOB(reI).PK →
JOB(rel).FK : → stands for "functionally determines")] \cup \{\text{sem(user)} = \emptyset\}

**Figure 25. Transform R10 - Ternary Nonunivocal Partial Regular Relationship Kind.**

**R10 - Syntactics:**

\[
\text{syn: } \{\{	ext{MANUFACTUR} \cdot \text{rel}.\{C\}, \text{CUSTOMER} \cdot \text{rel}.\{C\}, \text{MIDDLEMAN} \cdot \text{rel}.\{C\}, \text{LEASING} \cdot \text{rel}.\{C\}\}: \langle \text{MANUFACTUR} \cdot \text{con}.\{A\} \Rightarrow \text{MANUFACTUR} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{CUSTOMER} \cdot \text{con}.\{A\} \Rightarrow \text{CUSTOMER} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{MIDDLEMAN} \cdot \text{con}.\{A\} \Rightarrow \text{MIDDLEMAN} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{LEASING} \cdot \text{con}.\{A\} \Rightarrow \text{LEASING} \cdot \text{rel}.\{C\}\rangle \rangle \cup \{\text{syn(user)} = (\emptyset \lor \text{feedback from sem(user)})\}
\]

**R10 - Semantics:**

\[
\text{sem: } \{\{	ext{MANUFACTUR} \cdot \text{rel}.\{C\}, \text{CUSTOMER} \cdot \text{rel}.\{C\}, \text{MIDDLEMAN} \cdot \text{rel}.\{C\}, \text{LEASING} \cdot \text{rel}.\{C\}\}: \langle \text{MANUFACTUR} \cdot \text{con}.\{A\} \equiv \text{MANUFACTUR} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{CUSTOMER} \cdot \text{con}.\{A\} \equiv \text{CUSTOMER} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{MIDDLEMAN} \cdot \text{con}.\{A\} \equiv \text{MIDDLEMAN} \cdot \text{rel}.\{C\}\rangle \wedge \langle (\text{MANUFACTUR} \cdot \text{con}.\{A\} \cup \text{CUSTOMER} \cdot \text{con}.\{A\} \cup \text{MIDDLEMAN} \cdot \text{con}.\{A\}) \equiv \text{LEASING} \cdot \text{rel}.\{C\}\rangle \wedge \langle (\text{MANUFACTUR} \cdot \text{con}.\{A\} \equiv \text{MANUFACTUR} \cdot \text{rel}.\{C\}\rangle \wedge \langle \text{CUSTOMER} \cdot \text{con}.\{A\} \equiv \text{CUSTOMER} \cdot \text{rel}.\{C\}\rangle \wedge \langle (\text{MIDDLEMAN} \cdot \text{con}.\{A\} \equiv \text{MIDDLEMAN} \cdot \text{rel}.\{C\}\rangle \wedge \langle (\text{MANUFACTUR} \cdot \text{con}.\{A\} \cup \text{CUSTOMER} \cdot \text{con}.\{A\} \cup \text{MIDDLEMAN} \cdot \text{con}.\{A\}) \equiv \text{LEASING} \cdot \text{rel}.\{C\}\rangle \rangle \cup \{\langle \text{LEASING} \cdot \text{rel}.\{C\}: \langle \emptyset \lor \text{(relationship relation LEASING in 4NF? \Rightarrow determine multivalued dependencies and decompose LEASING manually)}\rangle\}
\]
6. CAD TOOL AND DERIVATION EXAMPLE

A prototype version of the CAD tool to derive network database structures has been implemented in Macintosh Pascal. It requires Macintosh 512K or above. The tool consists of two sets of programs - (1) to execute the required transforms, to display on a screen the currently involved portions of conceptual and logical structures, and to optionally save the derived logical structure in a disk file; (2) to print from the logical structure file a formatted and scaled diagram of the logical design. The third set of programs was developed to facilitate the validation of the derivation tool. This is the conceptual structure editor, that enables a rapid construction of any conceptual structure which is to be subjected to a logical conversion. Obviously, the structure can be saved, modified, or printed out at any stage. (Incidentally, all figures in this paper that address conceptual and network designs were prepared by storing the screen snapshots of the CAD session as MacPaint documents, and then printing them on a Laser Printer.)

Figure 26. Initial Conceptual Design.

The CAD prototype was designed to follow the user interface standards provided by Apple for software developers. This means extensive use of pull-down menus, the mouse, and the graphics routines stored in the ToolBox in ROM and accessible from within Pascal by QuickDraw calls. However, using the interpretative Macintosh Pascal rather than the Pascal compiler of the
Lisa development system led to some implementation difficulties on one hand, and to some innovative programming and a few inventions of our own on another hand. It is our intention to use Lisa Pascal for the final version of the CAD tool for the network derivation.

The derivation of relational structures has not been implemented yet. We are currently working on the programs specification. It is likely that the implementation of this part of our integrated CAD workbench will be attempted by means of Consula C compiler.

For a more comprehensive illustration of the described derivation system we chose a slightly modified version of the university database conceptual design, which was derived (at the earlier design phase) from the predefined set of user functions and presented in Maciaszek (1986). Figure 26 shows the printout of the initial conceptual design. Figure 27 represents a screen image taken during deriving the network logical structure. Figure 28 displays the scaled diagram printout of the derived network structure. Figure 29 presents an anticipated output from the CAD system for derivation of relational structures.

Figure 27. Screen Snapshot of the Network Derivation Process.
7. CONCLUSION

The methodology and tool for derivation of network and relational database schemas has been reported. Though being an outgrowth of the integrated database design workbench, the tool is self-contained. Because of the high-quality graphics resolution provided by the Macintosh, the tool seems to be attractive as a potential market product. It is user-friendly, yet incorporates the whole range of extensions to the basic entity relationship model and obeys the latest standardization proposals for both the network and relational model. We believe that the tool, once integrated with the overall design workbench, should prove useful practically and stimulative theoretically.

The prototype version of the system for derivation of network structures has been completed. A production version is expected to be available in the nearest future. The detailed program specification for derivation of relational structures is being defined. As being meant to be a professional tool, the system for both network and relational derivation is capable to recognize and transform as many as one hundred basic conceptual structures. To the best of our knowledge, this is the most comprehensive approach reported in the literature. Though some of the basic transforms are quite straightforward, some others are complex to the extent that the user's interaction is invaluable. Altogether, when considered in the integrated environment the derivation process gets increasingly complex and the best way to deal with this complexity is by means of the tool exhibiting the properties of an expert system. We believe that our approach constitutes an important step in this direction. At this juncture, a caveat. As our approach represents a direct response to the problem of practical and global logical design we were avoiding oversimplifying assumptions that usually underlie the research in this area. It was not our intention to optimize the derivation procedures (the problem, as stated, is NP-hard). We claim, however, that the derived structures are feasible (or even suboptimal from the viewpoint of user requirements on data) and can become very effective, especially if a performance tuning on them is applied.
Figure 29. Derived Relational Structure.

REFERENCES

BACHMAN, C.W. (1977): The Role Concept in Data Models, Proc. 3rd Int. Conf. on very Large Data Bases, Tokyo, Japan, pp.464-476.


DAHL, R. and BUBENKO, J.A. (1982): IDBD An Interactive Design Tool for CODASYL-DBTG-Type Data Bases, Proc. 8th Int. Conf. on Very Large Data Bases, Mexico City, Mexico, pp.108-121.


