Enhancing the Quality of Conceptual Database Specifications through Validation

Andreas Zamperoni

Leiden University
Dept. of Computer Science
Niels Bohrweg 2, NL-2333 CA Leiden
The Netherlands
Tel.: ++31/71/27 7103
Fax.: ++31/71/27 6985
email: zamper@wi.leidenuniv.nl

Perdita Löhr-Richter

Technical University Braunschweig
Dept. of Computer Science
Gaußstr. 12, D-3300 Braunschweig
Germany
Tel.: ++49/531/391 7447
Fax.: ++49/531/391 3298
email: loehr@idb.cs.tu-bs.de

Abstract

In this article, we present a validation approach and method to support the development of database applications. We explain how validation can cope both with the need for a formalized evaluation of correctness as well as the need for prototyping of conceptual database schemata. We define different levels of correctness for such schemata and show how these levels can be achieved through validation methods. Furthermore, we describe how our validation framework can be applied to other conceptual specifications.

Keywords: consistency, correctness, database application, database schema, database specification, extended ER model, test data, validation

1 Introduction

One of the main questions for the development of database specifications is:

> *How can we ensure that our specification is correct and reflects the desired portion of reality?*

In times of increasingly complex specifications of database applications, we need well-structured approaches to handle this complexity and guarantee qualitative good design results. One indispensable qualitative property of database specifications is correctness. To formally proof this property becomes soon very complex and may even not be possible. Thus, for the ordinary development work, a pragmatic approach should be on hand to complement (or even replace) the formal method: **validation**. In general, it is not possible to decide whether a specification is definitely correct via validation. Instead, the correctness property means to minimize the rate of errors within a specification. For this purpose, validation methods offer good error detection and diagnosis features which mainly bases on the generation of suitable test data. This test data can be reused for prototyping activities, measurements of specifications, and allows to involve users in early development stages. Due to these properties, validation suits as a pragmatic approach for the ordinary development work.

In the area of **software engineering**, testing is a well-known tradition and corresponding theories exist [OsCl 92, Be 91, Be 84]. These approaches focus on tests of programs or programming systems, i.e., dynamic structures. In this respect, most of them derive test data to test for functionality, coverage criteria, boundary cases, etc.

Information systems engineers to cope with different problems. The heart of information systems consists of complex database structures on top of which the applications' functionality is defined. Thus, the database structures must be shown correct and consistent before the corresponding functionality can be put on top. Validation approaches which work on this topic [NMLo 93, LeNo 90, DeDa 89, BrMa 86, No 83] agree on the underlying idea to populate the database specifications with (reasonable) data. By this, they try to show the database specification to be consistent which in general is an undecidable property [Ma 74]. Additionally, they offer the generated data to support prototyping of the database specification and, thus, illustrate the structure of the information and significant details. We generally agree with this point of view but think it necessary to adapt the validation technique for the complex **semantic data models** of today. Such data models belong to early development phases and, hence, include powerful high-level modeling concepts to ease specification tasks. This powerful expressiveness of the data models as well as the growing size of database specifications make it more difficult to judge whether a specification is consistent and offers an adequate solution to the modeling problem, emphasizing the need for an improved validation support.

In this paper, we propose a **validation approach for database specifications** of the early development stages, i.e., the conceptual design phase. Our approach comprehends a well-structured validation strategy and a data model centered algorithm for test data generation. The paper is organized as follows. In the next section, we introduce our basic validation strategy and show how to adapt it to conceptual database specifications. In section 3, we give a brief look at the objects to be validated: conceptual database specifications in terms of an extended ER model (EER). In section 4, we adapt and refine common correctness criteria for database specifications. We use them as goals and milestones for our validation approach which we present in section 5. The paper concludes with results concerning the decidability of the correctness criteria we used and gives an outlook of how to make our approach applicable for other specification techniques, i.e., different semantic data models or static parts of object models.

2 Validation strategy

Experiences in the field of validation and testing show that a lot of testing effort is done in an unstructured, ad hoc manner. Although, such proceeding may suit for small specifications, it is inadequate for complex and large specifications. Thus, the validation effort itself has to be structured and defined in advance (cf. fig. 1).

Structural frame

To start a validation effort, we have to formulate goals for it, i.e., express the (special purpose) correctness criteria we want to achieve. These criteria have to be defined as a set of **hypotheses** which reflects presumptions about the correctness of the considered specification. These hypotheses remain constant during the validation procedure and determine the results which we expect at its end. The next step is to execute a **testing method**. This method should be planned to support or to reject the chosen hypotheses. After the execution of the testing method, its results are compared with the expected results. Through such an **evaluation**, we decide whether we achieved our goals and can terminate the validation procedure or have to continue because the goals have not been achieved. In the latter case, we detect and classify **characteristics** of the failure in the delivered results. This classification leads to three different conclusions:

- The specification is not correct wrt. our set of hypotheses. Thus, we have to **modify the specification**.

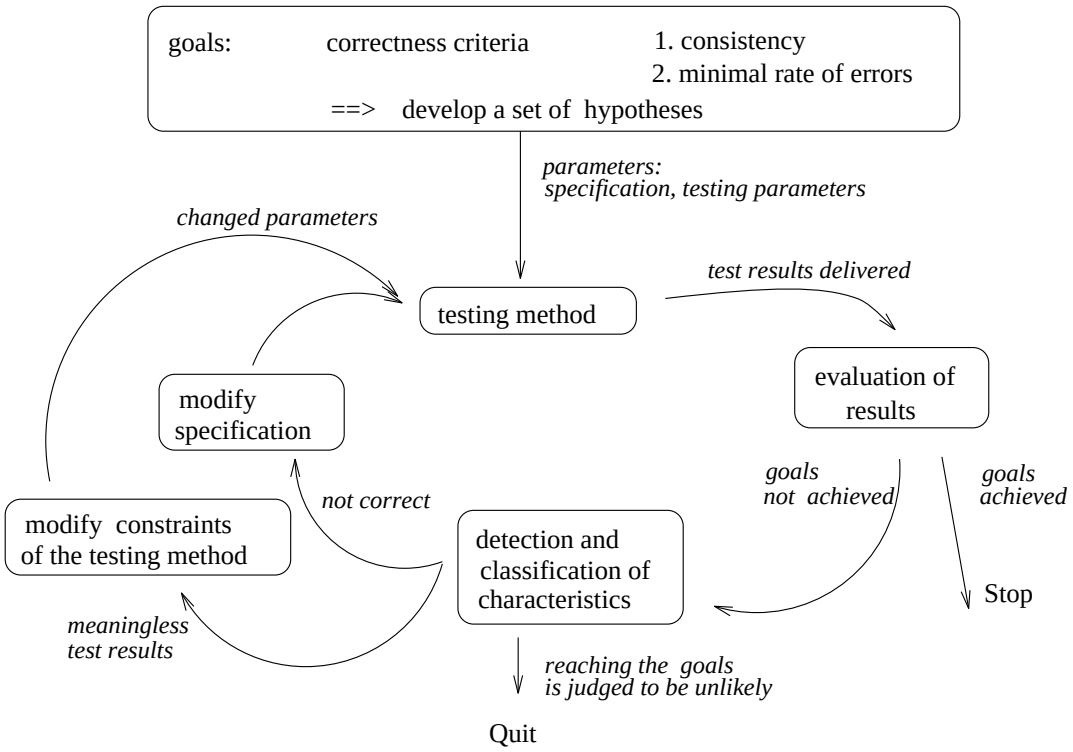

Figure 1: Structural frame for validation effort

- The delivered results are insufficient to evaluate our set of hypotheses, e.g., a hypothesis can neither supported nor rejected. Thus, the **constraints of the testing method** have to be modified.
- Reaching the goals, i.e., satisfying the hypotheses, is judged to be unlikely. In this case, the validation procedure should be aborted.

After modifying the incorrect or unsuitable parameters, we reiterate the validation procedure with the same set of hypotheses until it either stops (goals achieved) or is aborted (human judgement).

Incremental validation

Obviously, validation of database specifications is a complex task. Accordingly, it is necessary to break it up into smaller tasks. For this purpose, parts of the database specification must be identified which can be validated independently. To determine such parts, we use a simple property of conceptual database specifications: their **inherent dependency structure** [Lö 92]. This structure results from dependency relations between specifications parts. For example, a generalization relationship between two (or more) object classes induces a dependency of the generalized objects on the existence of the more specific objects. All sorts of relationships between specification objects can be transformed into dependencies [Lö 92]. We use these dependencies to split the complete validation task into incremental tasks and to define an execution order on all partial tasks (whether complex or incremental).

For this purpose, we introduce a graph structure, the so-called **dependency graph** [Lö 92], to which a specification can be transformed. The graph's nodes represent the specification elements and it's edges represent the relationship by which the specification elements depend on each other (cf. fig. 2).

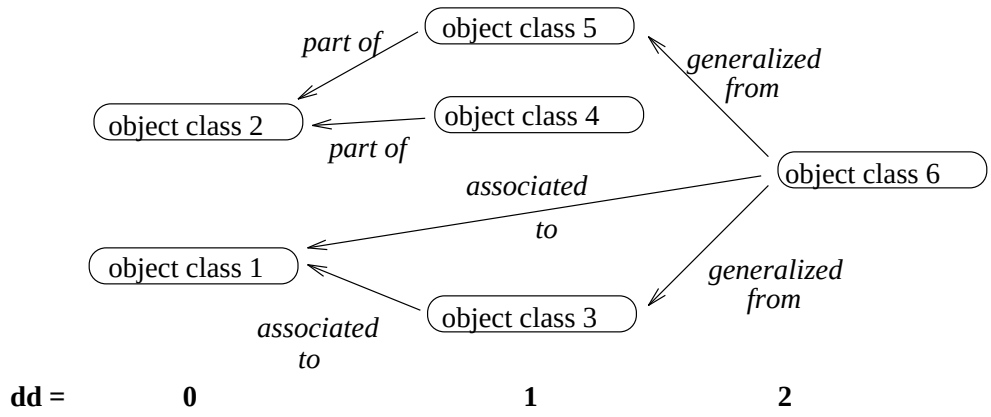

Figure 2: Partial dependency graph for database specifications

Such a graph determines a dependency degree (dd) for each object class within a specification[1]. The object classes without outgoing edges receive the $dd = 0$ which corresponds to independent object classes. The object classes depending onto those get the $dd = 1$ and so forth. Generally, an object class O receives the dd as follows:

$$dd_{\text{class O}} := \max\{dd \text{ of all object classes on which O directly depends}\} + 1$$

Now, it is easy to define an execution order for the incremental validation method[2]. It starts with the most independent elements of a specification (lowest dd). If these can be successfully validated the validation successively continues with the elements marked with the next higher dd. By this proceeding, we determine a well-structured and stepwise **execution order** of the validation process. In parallel, we ensure that already validated parts of a specification, i.e., parts with lower dd's, remain validated as long as errors only occur in more dependent parts of a specification, i.e., parts with higher dd's. This, in fact, enables us to proceed the validation task in an **incremental** manner, avoiding to evaluate again already validated objects.

3 Validation objects: conceptual specifications

The conceptual specification of a database application consists of a database schema, a set of integrity constraints, and a set of transactions. The schema (together with static integrity constraints) defines the static structure of the problem domain. On top of this structure, the behavior of the database application is described through transactions and dynamic integrity constraints. In turn, integrity constraints need a correct database schema, transactions need correct (dynamic) integrity constraints.

We use these global dependencies to subdivide entire database specifications into **validation subobjects**, namely the three parts of the specification. In turn, we employ the dependency structure within each of these validation subobjects to subdivide themselves into more refined validation subobjects.

We represent the various validation subobjects as nodes of a dependency graph (cf. fig.3). The edges express the validation dependencies between the elements of the graph, i.e., the validation subobjects. Explicitly, each edge indicates that the target node of the arrow has to be validated before the source node of the arrow. These dependencies imply the order of the validation process on the validation

[1] Note, that here $A \longrightarrow B$ means that A depends from B, i.e., A can be validated only if B has been validated.
[2] How to handle cyclic dependencies is shown in [Lö 92] and section 5

(sub)objects (cf. section 2). As mentioned in section 1, we focus on database schemata as the central component of database applications. Therefore, the validation subobject "database schema" is shown in more detail. To describe the schemata, we use an **extended ER model**.
We will give a brief overview over this specification technique in the following.

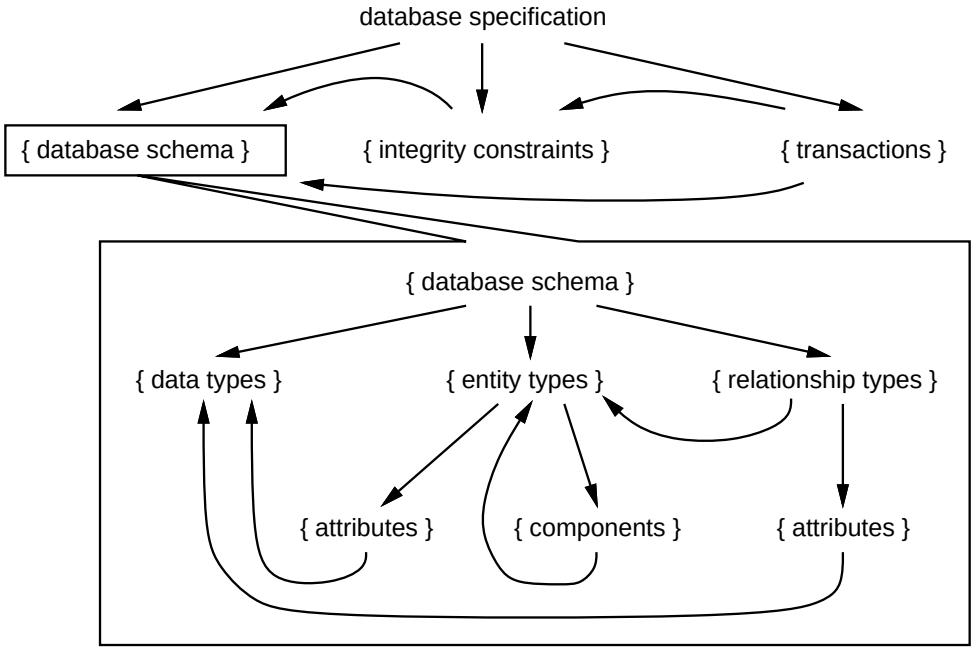

Figure 3: The dependency graph

A conceptual database schema: The EER schema

The EER model [EGHH+ 92] is an extension of Chen's ER model [Ch 76]. Due to the additional concepts and the intuitive graphical specification language, the EER schema especially supports the development of non-standard database applications (e.g. geoscientific databases). The main extensions to Chen's original ER model include:

multi, complex, and entity valued attributes: List, set, bag and record constructors are used to specify the range of attributes. In this way, abstract data types are built on top of already defined or existing data types. Entity valued attributes ("components") use an entity type as range and may as well be multi or complex valued. With this extension, association and aggregation of entities are modeled[3].

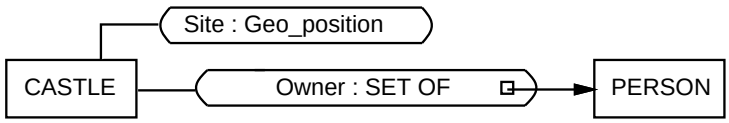

Figure 4: Attributes in the EER model

[3]The concept of entity valued attributes was introduced mainly for modeling purposes. Structurally it is equivalent to a ?:N-relationship type.

In figure 4, `Owner` is a set of `PERSON` entities, `Site` uses the abstract data type `Geo_Position` which could be a record of longitude, latitude, and height over sea level.

generalization and specialization of entities: For this purpose, the EER model provides the construct of a "type constructor". With a type constructor entities of "input types" can be grouped into new entity types, i.e., the "output types". For generalizations, we have many input types, i.e., the specific types, and one output type, i.e., the general type. For specializations, we have one input type, i.e., the general type, and many output types, i.e., the specific types. A type constructor can be partial or total. In case of a partial type constructor, not every entity of an input type needs to appear in one of the output types. For total type constructors holds that every entity of an input type has to appear in (exactly) one of the output types. Additional attributes can be defined for the output types, which in principle inherit the attributes of the input types.

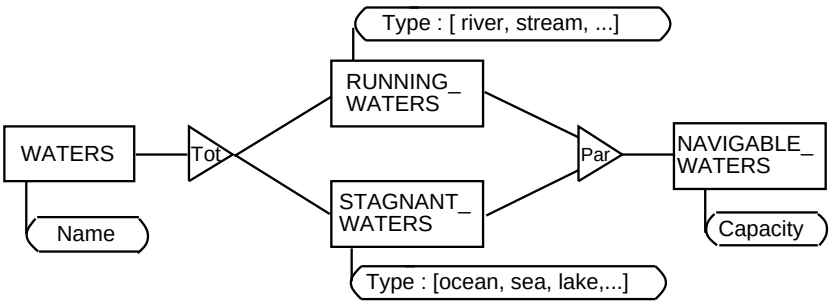

Figure 5: Type constructor in the EER model

Figure 5 gives an example for the use of type constructors. Input type `WATERS` is (totally) specialized to `RUNNING_WATERS` and `STAGNANT_WATERS`, adding a new attribute `Type` to both output types. From the specialized entity types a (partial) generalization `NAVIGABLE_WATERS` is then created, adding the attribute `Capacity` to the output type.

role concept for relationship types: Through different labeled roles, entity types can participate multiple times in the same relationship type (cf. fig. 6). By using roles, we distinguish different entities of the same entity type participating in one relationship.

cardinality intervals for relationship types: Cardinality intervals specify how many times an entity of an entity type may participate in relationships of a relationship type.

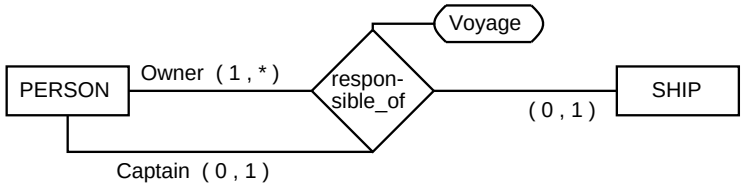

Figure 6: Relationship type in the EER model

In figure 6, an entity of type `PERSON` (i.e., a person) can be responsible for a ship through two different roles: as owner and/or as captain. The cardinality intervals determine that a person owns at least one ship and may own as much ships as he/she can afford. In the contrary, a person is only allowed for once (or not at all) to work on a ship as captain.

A detailed discussion of the concepts and semantics of the EER schema is provided in [EGHH+ 92].

4 Validation goals: correctness criteria

In the introduction we stated that the correctness of a specification is one of the most important questions posed in applications development. Often designers use an intuitive, experience based notion of a specification's correctness to decide to what level their work is correct. Nevertheless, a more **formal classification of "correctness"** is necessary to capture the levels of quality in the development process, and to serve as goals for goals for our validation process[4].

We distinguish three levels of correctness for database schemata:

> **Definition 1: Syntactical correctness**
> A database schema is syntactically correct if it doesn't violate syntactical and context-sensitive rules of the specification language used.

This level of correctness can be easily checked by a parser.

Although satisfying syntactical and context-sensitive rules, a schema might, of course, violate semantical constraints. The first step to approach semantical correctness is to detect contradictions in the schema, i.e., to check whether the schema is consistent. For this purpose, possible interpretations ("populations") of that schema have to be analyzed. Following logics, an interpretation of a specification is a collection of possible instances for every element of the specification (in case of EER schemata: entities for every entity type, relationships for every relationship type, etc.). The size of this collection, i.e., how many instances each element may have, is determined by the structure of the schema. Specification languages contain rules with which the population of a schema can be constraint. For example, cardinality intervals restrict the amount of possible instances of relationship types. We call these rules "**quantitative dependencies**". They numerically constrain the possible interpretations of a schema. To check satisfiability of quantitative dependencies only one of all possible interpretation has to be constructed, thereby proving that instantiation of the specification is - in principle - possible.

> **Definition 2: Consistency**
> A database schema is consistent, when at least one interpretation exists that contains finite, non-empty sets of allowed instances for each element of the database schema.

In the terminology of logics, such an interpretation is called "**model**". In logics, empty or infinite models exist, too. Nevertheless, these kind of models make no sense for database specifications. An empty or infinite model implies that the set of instances (for at least one specific element of that schema) is always empty or infinite. This is not meaningful for data to be stored in a database.

That syntactical correctness does not necessarily imply consistency, can be seen in figure 7.

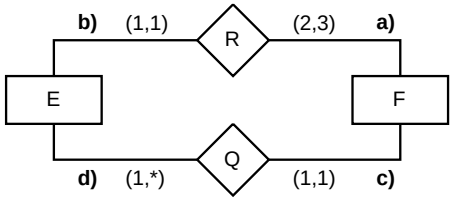

Figure 7: Example of an inconsistent EER schema

[4]Here, the term of "correctness" addresses one of the necessary quality criteria for specifications, as opposed to optional criteria like user-friendliness, portability, etc. as described in [GhJM 91].

Edge a implies that the number of instances of relationship type R is at least twice as high as the number of instances of entity type F. According to edge b, the number of instances of entity type E consequently has to be twice the number of F in order to fulfill the cardinality intervals of the relationship. That means, for each f of type F at least two and at most three e's of type E are needed to establish the demanded relationship R between E and F.
On the other hand, relationship Q (c and d) forces the number of instances of A to be lower or equal than the one of B. This happens because each a consumes at least one b and is allowed to consume more b's via the unrestricted participation in relationships of type Q. All conditions together lead to an inconsistent (although syntactically correct) specification.

Semantical correctness forms the highest level of correctness. It demands a database schema to reflect the desired portion of the problem domain. This level of correctness is, of course, a non-decidable quality [Ma 74] which has to be evaluated individually. This holds especially if the constraints of the problem domain have been specified only informally (as often the case).

> **Definition 3: Semantical correctness**
> A database schema is semantically correct in regard of the problem domain if the database schema is consistent and its (possible) interpretations are equivalent to the (possible) interpretations of the problem domain.

To check for semantical correctness involves generating a set of interpretations and "compare" them with the interpretations developers have in mind, as known from prototyping activities.

5 Validation method: idea and algorithm

According to definition 2, consistency of a database schema can be proved through the generation of (at least) one suitable interpretation. If we use our validation strategy presented in section 2 to show the consistency of a specification, we will choose as a hypothesis "an interpretation exists" and apply the method "generate an interpretation wrt. correctness level 2". Thereby, we do not only generate consistent populations of a database schema, but additionally are able to build meaningful test data collections for that schema. Thus, we deal with two different levels of correctness for our kind of database schema, i.e., EER schema, by:

1. offering a method to decide whether an EER schema is consistent or not
2. generating meaningful populations (test data) to support the evaluation of semantical correctness

These two aspects are discussed in the following.

Consistency through validation

The main question here is whether it it is possible to determine if the various **quantitative dependencies** within an individual EER schema allow in principle the generation of an interpretation or not. The quantitative dependencies we have to deal with arise from cardinality constraints between entity types and relationship types, grouping of entities through type constructions, and entity typed key attributes of our EER schema. Figure 8 provides an example for the different types of dependencies. Thus, the central problem of interpretation/test data generation comprises whether it is possible to fulfill the quantitative dependencies or not.
Our first approach to analyze quantitative dependencies is to draw up a linear inequality system for the schema. In this system, each inequality represents a quantitative dependency contained within

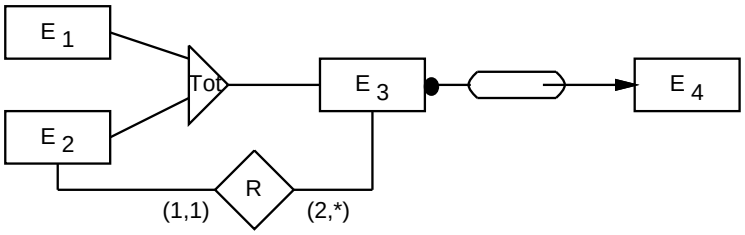

Figure 8: Example of quantitative dependencies in an EER schema. The "•" at E_3 indicates that E_4 is an (entity-valued) key attribute of E_3.

the schema. For our example in figure 8 holds (small letters are used for the number of instances of the schema elements labeled with capital letters):

$a)$	$r \geq 2 \cdot e_3$	cardinality constraint
$b) - c)$	$e_2 = r \quad (\Leftrightarrow e_2 \leq r \wedge e_2 \geq r)$	cardinality constraint
$d) - e)$	$e_1 + e_2 = e_3 \quad (\Leftrightarrow e_1 + e_2 \leq e_3 \wedge e_1 + e_2 \geq e_3)$	type construction
$f)$	$e_4 \geq e_3$	key attribute
$g) - k)$	$e_1, e_2, e_3, e_4, r \geq 1$	no empty sets of instances

The solution of this system of inequations describes the size of the population of the schema. In our example, in terms of inequalities, we have $r \geq 2 \cdot e_3 \overset{b)}{\Rightarrow} e_2 \geq 2 \cdot e_3 \overset{d)}{\Rightarrow} e_3 - e_1 \geq 2 \cdot e_3$ - a contradiction. No solution exists, and thereby we can easily show that this schema is not consistent.
We can observe that a "dependency cycle" exists in our example which causes that every element of that cycle quantitatively depends from itself. For example, E_3 refers to itself through the type constructor, $\mathtt{E}_2$, and `R`.

For those parts of a schema, which contain no such dependency cycles, it is always possible to find an (integer) solution for the number of instances for every schema element. This holds because in a non-cyclic path no contradictions in the ratios of the participating schema elements can occur. In terms of the corresponding system of inequations, this means that more variables than inequations exist (parametrized system) [PaSt 82], the related system of inequalities is always integer-solvable.
Special care must be taken for cyclic dependencies which cause the system of inequations to be (row) degenerated, i.e., more inequations than variables exist (cf. fig. 8: 5 variables, 11 inequations)[5].
In general, determining the existence of a (nonempty, finite) interpretation for an individual EER schema corresponds to solving the related system of inequations.
To draw up the system of inequalities for an EER schema, we apply the following transformation:

Definition 4: System of inequalities[6]

- for each cardinality constraint $E_i(P_k) \longrightarrow^{(x,y)} R_j$ with $x > 0$:

  $r_j \geq x \cdot e_i$
- for each cardinality constraint $E_i(P_k) \longrightarrow^{(x,y)} R_j$ with $y < \infty$:

  $r_j \leq y \cdot e_i$
- for each type construction with input classes $E_1, \ldots, E_n$ and output classes $E_{n+1}, \ldots, E_m$, $(m \geq n + 1)$:

[5] "row degenerated" refers to the coefficient matrix of the system of inequalities

[6] Notation: $E_i(P_k) \longrightarrow^{(x,y)} R_j$ means that an entity of E_i participates through role P_k at relationships of R_j at least x times and at most y times.

$$e_1 + \ldots + e_n \geq e_{n+1} + \ldots + e_m$$

- for each total type construction with input classes $E_1, \ldots, E_n$ and output classes $E_{n+1}, \ldots, E_m$, $(m \geq n+1)$ additionally:

  $$e_1 + \ldots + e_n \leq e_{n+1} + \ldots + e_m$$

- for each key component E_k (entity typed key attribute of an entity type E_i):

  $$e_k \geq e_i'$$

  If E_i has more than one key attribute, e_k, the number of required key components values for E_k decreases, because a combination of values is possible to build the key for an entity of E_i. Therefore, we write e_i', with $e_i' \leq e_i$. For how to calculate e_i', cf. [Za 92].

- for each E_i, R_j of the schema:

  $$e_i, r_j \geq 1$$

In [Za 92], we show that it is decidable whether a solution for such a system of inequalities exists. An equivalent proof for Chen's ER model was first presented in [LeNo 90]. We extended it to be applicable to systems of inequalities reflecting EER schemata. The proof uses the so-called "Ellipsoid-Algorithm" together with the algorithm "fractional dual" of [PaSt 82] to definitely decide about the existence of an integer solution for a system of linear inequations.
Through this proof, the consistency of the EER schema becomes a decidable quality.

Unfortunately, this proceeding is not suitable for practical use. The system of inequalities related to an individual EER schema becomes soon too complex to be processed with reasonable resources. Without optimization, solving the related system of inequations serves only to formally prove the decidability of consistency (for EER schemata).

So, we propose a more straightforward and intuitive approach to check quantitative dependencies. The basic idea is not to check the whole set of dependencies at once (via the system of inequalities) but to separate the complex analysis of the whole set into checking single dependency pathes, respectively dependency cycles. Therefore, we transform an EER schema into a so-called **quantitative dependency graph**. This graph is a subgraph of the dependency graph introduced in section 2 and can be derived from it. The quantitative dependency graph contains the same nodes as the dependency graph but less edges. It's nodes also represent schema elements (entity and relationship types). In contrast to the dependency graph, only those edges are considered that represent one of the dependencies listed in definition 4. The edges are labeled with the exact ratio of instances of the connected schema elements, or with a "$\leq/=$"-sign (in the case of a type construction or a key component - cf. fig. 9).
The cardinality ratios between the nodes of the quantitative dependency graph depend on the direction which is chosen to traverse a cycle, as indicated in figure 9. Note that it is arbitrary in which direction a cycle is traversed[7], the results concerning consistency are the same in all directions. By covering all possible pathes in the so created quantitative dependency graph, it is possible to calculate and keep track of all the cardinality ratios between schema elements, i.e., nodes. A cycle in the quantitative dependency graph indicates that each of the involved nodes (schema elements) recursively constraints its own number of instances through the chain of the other nodes (elements) of that cycle. We have to check ratios computed during the traversal of a cycle as well as ratios between input and output types of a type constructor. Hence critical, i.e., unsatisfiable, dependencies in the EER schema can be detected. In case of an unsatisfiable ratio (just like in figure 7), the related EER schema elements indicate where the specification has to be revised. If no "critical" cycle has been detected the EER schema is consistent.

[7] Sometimes, one direction of a cycle within a directed graph is addressed as "circuit".

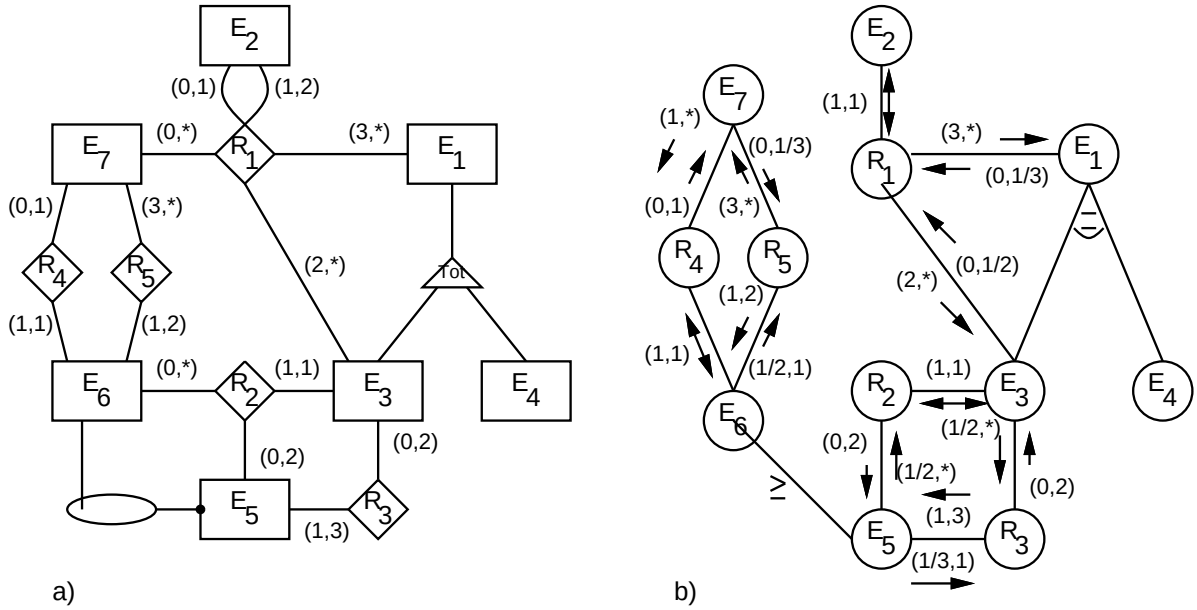

Figure 9: Example of a transformation from a) EER schema to the related b) quantitative dependency graph (with cardinality ratios)

Based on the quantitative dependency graph, we developed an algorithm which determines proposals for the size of populations of a consistent EER schema. After having checked consistency, the cycles are split up and a total order is imposed on all nodes of the quantitative dependency graph. With termination ensured by the previous ratio check, a backtracking algorithm assigns numbers of instances to each node, i.e., schema element, without violating the ratios imposed by the cardinality constraints. The assignment process can be varied by predefining desired intervals for the numbers of instances for each schema element [Za 92]. These proposals for the size of populations serve to trigger the generation of the sound test data.

Test data

The approach presented so far serves for two goals. First, it yields a terminating method to decide about the consistency of a given (EER) schema. Furthermore, it enhances the quality of the validation process itself. With the certainty of consistency and with an already determined size of the appropriate interpretation, the next level of correctness, the semantical correctness (cf. definition 3), can be tackled. The validation process as described in section 2 is simplified to the pure generation of "test instances" for each schema element. In order to accomplish this task, a framework for a test data generator which handles the quality parameters of test data generation (cf. [GhJM 91]) has been worked out and presented in [Za 92]. It incorporates the incremental validation strategy presented in section 2 and delivers, step by step, full-scale test data sets which can be evaluated by the designer. Advantage of this approach is that the designer can rely on the EER schema as comprehensible specification technique, and that his schema is used as **executable specification** to prototype his design.

6 Conclusions

In this paper, we proposed a validation method for conceptual database schemata. This method bases on a well-structured validation strategy with the following main parameters:

- the validation object: a conceptual database schema

- the validation goal: a set of hypotheses
- the testing method

To express the database schema, we used a semantic data model, the EER model. To define the goals, we considered traditional notions of correctness and refined them for database specifications.
Under these assumptions, we succeeded in showing that the (usually undecidable) correctness property "consistency" is decidable for EER schemata. This result provides the theoretical basis for us to develop an algorithm which decides whether an individual EER specification is consistent or not and, in case of consistency, delivers generated test data. Such data can be used to investigate semantical correctness via prototyping.

Semantic data models of today cover a similar set of modeling concepts [HuKi 89], namely aggregation, association, and generalization/specialization. Due to this fact, it is possible to show that consistency is decidable for other semantic data models, too. Additionally, we identified the same set of modeling concepts in object oriented specification languages. They are used to describe the static structure of objects and object classes respectively. Thus, we presume that consistency is decidable for the static structure of object oriented specifications, too. We supported our presumption by successfully applying our validation framework to the object oriented specification language TROLL [JSHa 91]. Further investigation of this topic will be part of our future work.

References

[Be 84] Beizer, B.: *Software System Testing and Quality Assurance.* Van Nostrand Reinhold Company, USA 1984

[Be 91] Bernot, G.: *Testing Against Formal Specifications: a Theoretical View.* Rapport de Recherche du Laboratoire d'Informatique, URA 1327 du CNRS, Departement de Mathematique ed d'Informatique, Ecole Normale Superieure, Paris, January 1991

[BrMa 86] Bry, F.; Manthey, R.: "Checking Consistency of Database Constraints: A Logical Basis", *Proc. 12th Int. Conf. Very Large Data Bases*, Kyoto, Japan 1986 (13–20)

[Ch 76] Chen, P.: "The Entity-Relationship Model - Toward a Unified View of Data" In: *ACM Transactions on Database Systems*, Vol.1, No.1, March 1976 (9–36)

[DeDa 89] Delcambre, L.; Davis, K.: "Automatic Validation of Object-Oriented Database Structures", *Proc. 5th Int. Conf. Data Engineering*, Los Angeles, California, IEEE Computer Society, 1989 (2–9)

[EGHH+ 92] Engels, G.; Gogolla, M.; Hohenstein, U.; Hülsmann, K.; Löhr-Richter, P.; Saake, G.; Ehrich, H.-D.: "Conceptual Modelling of Database Applications Using an Extended ER Model", In: *Data & Knowledge Engineering*, North-Holland, 9(2), 1992 (157–204)

[GhJM 91] Ghezzi, C.; Jazayeri, M.; Mandrioli, D.: *Fundamentals of Software Engineering.* Prentice-Hall International, USA 1991

[HuKi 89] Hull, R.; King, R.: "Semantic Database Modeling: Survey, Applications, and Research Issues", In: *ACM Computing Surveys*, Vol.19, No. 3, 1989 (201–260)

[JSHa 91] Jungclaus, R.; Saake, G.; Hartmann, T.: "Language Features for Object-Oriented Conceptual Modeling", *Proc. 10th Int. Conf. on Entity-Relationship Approach*, Teory, T. (ed.), San Mateo (CA), 1991 (309–324)

[NMLo 93] Neufeld, A.; Moerkotte, G.; Lockemann, D.: "Generating Consistent Test Data: Restricting the Search Space by a Generator Formula", In: *The VLDB Journal*, 2(2), 1993, (173–213)

[LeNo 90] Lenzerini, M.; Nobili, P.: "On the Satisfiability of Dependency Constraints in Entity-Relationship Schemata", In: *Information Systems*, Vol.15, No.4, 1990 (453–461)

[Lö 92] Löhr-Richter, P.: "Basic Units for the Database Design Process", Studer, R. (Hrsg.): *Proc. 2nd Workshop Informationssysteme und Künstliche Intelligenz: Modellierung*, Ulm, Feb. 1992, Informatik-Fachberichte 303, Springer-Verlag, 1992 (56–67)

[Ma 74] Manna, Z.: *Mathematical Theory of Computation.* Mc Graw - Hill, USA 1974

[No 83] Noble, H.: "The Automatic Generation of Test Data for a Relational Database", In: *Information Systems*, Vol.8, No.2, 1983 (79–86)

[OsCl 92] Osterweil, L.; Clark, L.A.: "A Proposed Testing and Analysis Research Initiative", In: *IEEE Software*, Vol.9, No.5, 1992 (89–96)

[PaSt 82] Papadimitriou, C. / Steiglitz, K.: *Combinatorial Optimization - Algorithms and Complexity.* Prentice-Hall, USA 1982

[WaFu 89] Wallace, D.R., Fuji, R.U.: "Software Verification and Validation: An Overview", In: *IEEE Software*, Vol.6, No.3, 1989 (10–17)

[Za 92] Zamperoni, A.: "A Conceptual Framework for the Generation of Logical Models for EER Schemata", Diploma Thesis (german), Technical University of Braunschweig, 1992