

IFIP TC 2 Working Conference on  
Formal Description of Programming Concepts—II  
Garmisch-Partenkirchen, FRG, 1-4 June 1982

Organized by  
IFIP Technical Committee 2  
Programming  
International Federation for Information Processing

Program Committee  
E. J. Neuhold (Chairman), J. W. de Bakker, D. Bjørner  
A. Blikle, C. Böhm, E. K. Blum, J. B. Dennis, J. Goguen  
S. Igarashi, P. Lauer, R. Milner, M. Nivat



NORTH-HOLLAND PUBLISHING COMPANY  
AMSTERDAM • NEW YORK • OXFORD

# FORMAL DESCRIPTION OF PROGRAMMING CONCEPTS – II

---

Proceedings of the IFIP Working Conference on  
Formal Description of Programming Concepts—II  
Garmisch-Partenkirchen, FRG, 1-4 June 1982

020

Edited by  
DINES BJØRNER  
*Technical University of Denmark  
Lyngby, Denmark*



1983

NORTH-HOLLAND PUBLISHING COMPANY  
AMSTERDAM • NEW YORK • OXFORD

© IFIP, 1983

*All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.*

ISBN: 0 444 86619 1

*Published by:*  
NORTH-HOLLAND PUBLISHING COMPANY  
AMSTERDAM • NEW YORK • OXFORD

*Sole distributors for the U.S.A. and Canada:*  
ELSEVIER SCIENCE PUBLISHING COMPANY, INC.  
52 Vanderbilt Avenue  
New York, N.Y. 10017

DR. HANS BEKIĆ  
1936 – 1982

PRINTED IN THE NETHERLANDS

## CONTENTS

|   |     |
|---|-----|
| Dedication  | v   |
| Preface   | xi  |
| <br>  |     |
| <u>Session 1: Chairman: Dines Bjørner, Tuesday, June 1, 1982</u>  |     |
| 1. H. Ganzinger   |     |
| <i>Denotational Semantics for Languages with Modules</i>          | 3   |
| Questions and Answers   | 21  |
| 2. M.C. Gaudel  |     |
| <i>Correctness Proof of Programming Language Translations</i>     | 25  |
| Questions and Answers   | 43  |
| 3. P. Mosses  |     |
| <i>Abstract Semantic Algebras</i>                                 | 45  |
| Questions and Answers   | 71  |
| 4. H. Christiansen and N. Jones                                   |     |
| <i>Control Flow Treatment in a Simple Semantics-Directed</i>      |     |
| <i>Compiler Generator</i>   | 73  |
| Questions and Answers   | 97  |
| <i>Session Discussion</i> Responder: P. Lucas                     | 99  |
| <br>  |     |
| <u>Session 2: Chairman: C. Böhm, Wednesday, June 2, 1982</u>      |     |
| 1. E.G. Wagner  |     |
| <i>Functorial Hierarchies of Functional Languages</i>             | 107 |
| Questions and Answers   | 122 |
| 2. M. Broy  |     |
| <i>Fixed Point Theory for Communication and Concurrency</i>       | 125 |
| Questions and Answers   | 147 |
| <i>Session Discussion</i> Responder: A. Blikle                    | 149 |
| <br>  |     |
| <u>Session 3: Chairman: J.B. Dennis, Wednesday, June 2, 1982</u>  |     |
| 1. R. Kuiper and W.P. de Roever                                   |     |
| <i>Fairness Assumptions for CSP in a Temporal Logic Framework</i> | 159 |
| Questions and Answers   | 168 |

|   |            |
|---|------------|
| 2. M. Broy and M. Wirsing<br><i>On the Algebraic Specification of Finitary Infinite<br/>Communicating Sequential Processes</i><br>Questions and Answers | 171<br>197 |
| 3. G.D. Plotkin<br><i>An Operational Semantics for CSP</i><br>Questions and Answers   | 199<br>224 |
| 4. M.C.B. Hennessy and W. Li<br><i>Translating a Subset of Ada into CCS</i><br>Questions and Answers  | 227<br>248 |
| <i>Session Discussion</i> Responder: J. de Bakker   | 251        |

Session 4: Chairman: M. Paul, Thursday, June 3, 1982

|  |            |
|--|------------|
| 1. A. Maggiolo-Schettini and J. Winkowski<br><i>Towards a Programming Language for Manipulating<br/>Relational Data Bases</i><br>Questions and Answers | 265<br>278 |
| <i>Session Discussion</i> Responder: E. Neuhold  | 281        |

Session 5: Chairman: J.B. Dennis, Friday, June 4, 1982

|   |            |
|---|------------|
| 1. J.A. Bergstra and J.W. Klop<br><i>Formal Proof Systems for Program Equivalence</i><br>Questions and Answers          | 289<br>303 |
| 2. P. Padawitz<br><i>Equational Data Type Specifications and<br/>Recursive Program Schemes</i><br>Questions and Answers | 305<br>329 |
| 3. J.-P. Jouannaud, P. Lescanne, and F. Reinig<br><i>Recursive Decomposition Ordering</i>                               | 331        |
| <i>Session Discussion</i> Responder: G. Cousineau   | 349        |

Session 6: Chairman: P. Lauer, Friday, June 4, 1982

|   |            |
|---|------------|
| 1. H.J. Genrich and P.S. Thiagarajan<br><i>Well-Formed Flow Charts for Concurrent Programming</i><br>Questions and Answers                            | 357<br>381 |
| 2. I. Castellani, P. Franceschi, and U. Montanari<br><i>Labeled Event Structures: A Model for Observable<br/>Concurrency</i><br>Questions and Answers | 383<br>400 |
| <i>Session Discussion</i> Responder: G. Cousineau   | 401        |

Accepted Papers Not Presented

|  |     |
|--|-----|
| M.A. Casanova, J.M.V. de Castilho, and A.L. Furtado<br><i>Properties of Conceptual and External Database Schemas</i> | 409 |
| E. Best<br><i>Relational Semantics of Concurrent Programs<br/>(With Some Applications)</i>                           | 431 |
| List of Participants   | 453 |
| Author Index   | 455 |

PROPERTIES OF CONCEPTUAL AND EXTERNAL  
DATABASE SCHEMAS

M.A. Casanova                      J.M.V. de Castilho\*                      A.L. Furtado  
Pontifícia Universidade Católica do Rio de Janeiro  
Rua Marquês de São Vicente, 225  
22453 - Rio de Janeiro, RJ  
Brasil

The design of database applications is investigated by identifying a number of properties that the conceptual and external schemas must satisfy. The list of properties covers, among others, basic issues in database design, such as consistency of the set of integrity constraints, the design of schemas with built-in operations, the interaction between the conceptual and external schemas and the interplay between several external schemas. The definition of each property, with some restrictions, may be expressed within one of the three formalisms: first-order predicate calculus, dynamic logic and modal logic. The relational model is adopted throughout the development.

INTRODUCTION

This paper addresses the design of database applications within the framework proposed by the ANSI/SPARC Study Group of DBMSs [2]. The ANSI/SPARC proposal divides the database description into three levels: the conceptual schema, describing the enterprise as a whole; the internal schema, describing the physical organization of the database; and the external schemas, describing the way each group of users sees the enterprise. (For alternative structures see [27, 34]).

The basic goal of this paper is to investigate under what conditions the design of the conceptual and external schemas of a database can be considered adequate. This goal is achieved by formally defining a series of properties that the various schemas must satisfy. As a consequence, the relationship between database design and database theory is clarified, and results in database theory are assessed from the design perspective.

Included in the discussion are schemas that contain a predefined repertoire of updates. This extension is both natural and important since it accounts for a large number of "closed" or "menu" database applications, such as airline and hotel reservation, point-of-sale inventory control, electronic banking, and others. Even more important is the fact that it can also be used with "open" database applications where users are still allowed to write full-fledged programs, except that the programs are limited to modify the database only by invoking updates from the predefined repertoire. This is a convenient strategy to enforce integrity constraints, quite in the spirit of the encapsulation strategy from the abstract data type area [29]. In the context of databases, built-in operations were considered, for example, in [7].

This paper is based largely on four formalisms. As far as database concepts are concerned, the relational model of data [14] is adopted. The schema design properties considered are formalized within First-Order Logic, if they do not involve updates, or Dynamic Logic [10, 11, 26], otherwise. Modal Logic [13] is a

\* On leave from the Universidade Federal do Rio Grande do Sul

used when dynamic consistency criteria are considered. A survey of the relationship between these formalisms (except Modal Logic) appears in [22].

The literature on schema design theory is reasonably extensive, especially for the relational model. Therefore, other references to related work are delayed to Section 2 when each specific property is described.

The paper is organized as follows. Section 2 contains an informal presentation of the schema design properties discussed. Section 3 formalizes the notions of conceptual and external schemas and their basic properties. Only concepts from First-Order Logic are used in this section. Section 4 repeats the discussion in Section 3 for schemas containing a predefined set of updates. It depends largely on Dynamic Logic. Section 5 formalizes the concept of dynamic consistency criteria using Modal Logic. Finally, Section 6 contains conclusions and directions for future research.

## 2. AN OVERVIEW OF SCHEMA DESIGN CONCEPTS

In this section we informally describe the basic schema design concepts and properties that will be formalized later. For ease of reference, properties will be numbered and grouped in Figure 2.1 at the end of the section.

### 2.1 - Basic Concepts

We discuss in this section how a database is described.

A conceptual schema describes the database as a whole. It consists of a set of data structures describing how data is organized in the database and a set of consistency criteria specifying the allowed data values. In the relational model of data, the basic data structures are flat tables, and the consistency criteria can be described as first-order formulas. A language used to describe schemas is called a data definition language.

A set of data values satisfying the consistency criteria is called a consistent or valid database state.

An external schema describes those aspects of the database that are meaningful to a group of users. Just like a conceptual schema, it contains data structure descriptions and consistency criteria. We may also talk here of consistent or valid database states.

We now state a basic property any schema (conceptual or external) must satisfy:

P1: Consistency: The set of consistency criteria must not contain contradictions or, equivalently, at least one consistent state must exist.

Property P1 is, in general, very hard to check. However, there are important special cases where this property is trivially satisfied. In the relational model, almost all classes of consistency criteria considered in the literature [1, 5, 14, 18, 19, 23, 30, 32, 41, 42, 46, 47] are special cases of the extended embedded implicational dependencies (XEIDs) [19]. But a set of XEIDs is trivially satisfied by the "initial" state, that is, by the state where all relations are empty. Therefore, if all consistency criteria are XEIDs, then Property P1 is trivially satisfied.

The following property would also be desirable (although not necessary):

P2: Logical Independence: no consistency criterion is logically implied by the others.

Again, checking this property becomes easy in certain special cases. For example, if all criteria are functional or multivalued dependencies [18, 47], there is an

efficient algorithm to check logical independence [38] (for surveys about other aspects of the design of relational databases that have also received considerable attention see [4, 15, 44]).

We conclude this section by observing that the schema definition may also include dynamic consistency criteria imposing restrictions directly on the state transition that can be brought about by updates. Examples are: (1) employees' salaries never decrease; (2) once an employee is fired, he can never be readmitted. Obviously, dynamic consistency criteria, such as those above, would be enforced with the help of auxiliary structures (i.e., names of former employees) summarizing information about past states of the database.

Regarding dynamic consistency criteria, we introduce the following property:

P3: dynamic consistency: no state transition (resulting from a sequence of one or more updates) violates any dynamic consistency criteria.

### 2.2 - Dynamic Aspects of Database Design

By contrast with Section 2.1, we concentrate in this section on the problem of defining how the database can be modified. We assume that we are given a data manipulation language (DML) which can be used to write programs, or updates, for this purpose. We will be concerned with two problems:

- (1) What updates are allowed?
- (2) Is the DML chosen in some sense adequate?

As for the first problem, we consider that an update  $u$  is valid, or preserves consistency, iff  $u$  maps the set of consistent database states into itself. This basic requirement on updates was studied in [10, 11, 20, 24].

The second problem requires a longer discussion. We first observe that the database designer may want to specify that certain updates should be available to the users. The availability of updates may be expressed as pairs of formulas  $(P, Q)$  that we call update specifications ( $P$  and  $Q$  are called pre - and post-conditions, respectively).

Update specifications impose a restriction on the DML, which we state as a new property. We say that a program  $p$  achieves  $(P, Q)$  iff  $p$  maps every state satisfying  $P$  into some state satisfying  $Q$ .

P4: Adequacy of the Schema DML: every update specification must be achievable by some valid update (written in the DML chosen).

Property P4 should be understood as follows. During an early stage, the designer anticipates that certain changes in the database will have to be made. He then describes these changes non-procedurally using pairs of pre - and post-conditions. Later, he chooses a schema DML. Finally, he checks if the DML chosen is powerful enough to achieve every anticipated state transition.

Pairs of formulas may also be used to specify how a (static) consistency criterion is to be enforced. For example, consider constraint "every employee must work for some department" and the update "delete department  $d$ ". Then, the update can be implemented either by deleting  $d$  and all employees working in  $d$ , or by deleting  $d$  only if no employee works in  $d$ . Note that both methods satisfy the criterion, but they are widely different. The structural model of [45] indeed has two different types of constraints corresponding to the two interpretations above.

Following the spirit of Property P4, we have two other properties. First, it may be desirable to reach any consistent database state from any other consistent database state. This property is related to the notion of completeness in [39].

P5. Completeness of the Schema DML: given any pair (A,B) of consistent database states, there must be some valid update (of the DML) that maps A into B.

Second, we may identify some particular consistent state  $A_0$  as the "empty" or initial state. Then, we may require that:

P6. Reachability: given any consistent database state A, there must be some valid update that maps  $A_0$  into A.

We close this section with a few comments on the concepts introduced here. First, the notion of valid update amounts to forcing a program to satisfy certain pre- and post-conditions. Thus, the techniques developed for constructing correct software can be used here. However, Properties P4, P5 and P6 impose restrictions not on programs, but on the DML chosen since they assert that it must be possible to construct updates with certain characteristics. Hence, they may be very difficult to check, unless the DML was designed so that it trivially satisfies these properties [39].

### 2.3 - Schemas with Built-in Operations

The schema DML is normally understood as the basic DML, that is, the DML of the data model or the DML supported by the DBMS. However, this approach makes it difficult to assure that every update is valid and, hence, it compromises the quality of data. To circumvent this problem, one may: (i) provide the schema (conceptual and external) with built-in operations to modify the database (written in the basic DML) that preserve consistency; (ii) force users to utilize the built-in operations when they modify the database. This strategy then obviously guarantees that all updates preserve consistency. By the restricted schema DML, we mean the set of programs (of the basic DML) that modify the database only through the built-in operations.

In addition to the properties of Section 2.2, we may list certain other requirements on a restricted schema DML. Given a schema with built-in operations, we require:

P7. Consistency preservation: each built-in operation is valid, i.e., preserves consistency.

P8. Operation independence: no built-in operation can be eliminated without affecting the set of all possible database state transitions.

P9. Operation applicability: for each built-in operation there is at least one consistent state where the operation can be successfully applied to modify the database.

We close with some observations about the concepts introduced here. First, although built-in operations are an excellent mechanism to assure that every update is valid, they may raise serious problems when one considers properties P4, P5 and P6. For example, it may not be clear if, given set of built-in operations and a pair (A,B) of consistent database states, it is possible to write a program in the restricted DML that takes A to B. Second, some of the properties discussed in Sections 2.2 and 2.3 may not make sense when we consider external schemas with built-in operations, although they all apply in principle to conceptual and external schemas. For example, it is quite possible that the set of all consistent database states of an external schema E contains more states than the set reachable from the initial state by repeatedly applying built-in operations of E. This follows because, for security reasons, some consistent state A of E may not be reached by applying only updates available through E, but A can indeed be reached indirectly as a result of the combined update activity of other external schemas of the same database. Finally, we observe that sometimes external users may not be able to observe completely the effect of an update. Consider the case where an external user sees only information about employees whose salaries do not exceed a certain amount M, but he is able to

raise salaries beyond this limit. The effect of raising someone's salary beyond the limit will look to the external user as a delete operation, since the employee will disappear from his external schema. It is also interesting to note that if the external user had among his update operations a "delete employee" update, then deleting an employee or raising his salary beyond the limit would have the same external effect.

### 2.4 - The Interaction between the Conceptual and External Schemas

One important fact concerning external schemas is that an (external) state is a virtual object with no independent existence. The data an external user wants to see is materialized from a (conceptual) database state via a function which is part of the external schema definition. Likewise, each operation of an external schema is always translated into a program in the conceptual DML.

With this in mind, two other properties become important.

P10. translation correctness: the translation  $t$  of an operation  $u$  is correct in the sense that the result of  $u$  coincides with the external state constructed from the result of  $t$ .

This property is the minimum we require from  $t$ . Further requirements are discussed in [3, 8, 16, 21, 33, 35], which concentrate specifically on the view update problem.

P11. conceptual consistency preservation: the translation of each operation maps the set of consistent database states into itself.

If we allow the translation of each operation to modify data in the database only through conceptual schema built-in operations then Property P7 implies Property P11. Hence, this restriction should be adopted as much as possible since it greatly simplifies database design.

All other properties we consider in this section depend on the design strategy one has in mind. Suppose first that we adopt the position that the conceptual schema must contain a complete description of the application and that each external schema is just a window through which a group of users sees the database. This implies that each external schema does not contain any restriction that may not be deduced from those in the conceptual schema. (Note that the presence of consistency criteria in the external schemas is still necessary so that each user gets an accurate description of the data he sees or manipulates).

More precisely, each external schema must also satisfy the following property, which was studied in [9, 28].

P12. logical dependence: each external consistency criterion is a logical consequence of the conceptual consistency criteria in the sense that any external state constructed from a consistent database state is also consistent (with respect to the external consistency criteria).

Therefore, if all external schemas satisfy P11 and P12, then no operation on one external schema can produce as a side-effect an inconsistent external state of another schema.

Failure to satisfy property P12 can have two interrelated consequences. First, an inconsistent external state may be generated from a consistent database state. Second, an update to an external schema may produce as a side-effect an inconsistent external state of another external schema (this is really a consequence of the previous remark).

However, Property P12 may impose excessive restrictions on external schemas, if we

adopt a different design strategy. We may consider the conceptual schema as a community description of the application, incorporating just those properties that are stable and common to all users, such as legal restrictions or long-range, global policies. Each external schema inherits these properties from the conceptual schema, but it also reflects the peculiarities of a group of users and short-term policies that are subjected to change. Hence, Property P12 would be too strong in this case.

This second design strategy requires a change of attitude towards the role of the conceptual schema. Instead of considering it as the central concept, we take the active schema, defined as the conceptual schema augmented with all consistency criteria derived (in a precise sense to be described in Section 3.2) from the definition of the external schemas. When deriving the active schema it may be easier to proceed iteratively, starting from the conceptual schema and then adding the external schemas one by one. Similarly, during the lifetime of a database, new external schemas can be added or existing ones dropped. The separate study of a single external schema together with the conceptual schema may also be useful to isolate contradictions between characteristics of a group of users and global policies embodied in the conceptual consistency criteria.

All concepts and properties developed previously for conceptual schemas also apply to active schemas. In particular, a database state is now valid or consistent iff it satisfies all constraints of the active schema; likewise, an update is now valid iff it maps the set of consistent database states (in the above sense) into itself. Furthermore, by construction, the constraints of the active schema logically imply all external consistency criteria. Hence, each consistent database state (of the active schema) generates consistent external states and Property P12 is obtained by a fiat.

The two design alternatives compare as follows. The first design alternative certainly facilitates the definition of external schemas since all restrictions of the application are already embodied in the conceptual schema. It also simplifies checking consistency preservation, as compared to the second design alternative, which requires checking preservation of the potentially redundant criteria of the active schema. On the other hand, a conceptual schema designed according to the first strategy is less stable and, hence, more difficult to maintain. Or, in different words, if we follow the second design alternative, moderate evolutions in the application will impact just the external schemas and, hence, the active schema, while keeping the conceptual schema stable. Evolvability is enhanced if the conceptual schema DML has the completeness property (P5), since no built-in operation of an external schema will be left untranslated by lack of flexibility of the schema DML. But to achieve full evolvability one should closely control the effects of a change in one external schema over the active schema and, consequently, over other external schemas. If a change on the set of consistency criteria of an external schema E makes the corresponding active schema inconsistent, then consistency should be restored, either by conveniently changing the constraints of E again, or the constraints of some other external schema, or even the constraints of the conceptual schema.

## 2.5 - The Interaction between External Schemas

It is worth investigating not only the interaction between an external schema and the underlying conceptual schema, but also the interplay between external schemas over the same conceptual schema. We already mentioned in Section 2.4 that, if Properties P11 and P12 are met, then no operation on one external schema can produce as a side-effect an inconsistent external state of another schema. But this does not say that there is no side-effect. So, it might be of interest to detect which, and how, operations of an external schema affect other external schemas.

Given two external schemas E and E', we may classify the effects of an operation  $\bar{o}$  of E on E' as either an influence on queries or an influence on operations. By an

influence on queries, we mean that data items visible through E' are changed as a result of applying  $\bar{o}$ ; by an influence on operations, we mean that there is an operation  $\bar{o}'$  of E' whose result depends on whether or not  $\bar{o}$  was applied.

Let us consider first the problem of detecting if an operation  $\bar{o}$  of E influences some query of E'. Note that  $\bar{o}$  and E' have no direct relationship and, in fact, different languages are in principle used to describe  $\bar{o}$  and E'. But both  $\bar{o}$  and E' are virtual objects in the sense that  $\bar{o}$  is implemented in terms of a program t at the conceptual schema level and E' is likewise defined in terms of the conceptual schema. Hence, the problem of checking if  $\bar{o}$  influences some query of E' reduces to testing if t affects the definition of E' (since this definition can be thought as the translation of a query that accesses all data in the external schema).

The problem of detecting if an operation  $\bar{o}$  of E influences an operation  $\bar{o}'$  of E' is again treated by considering the translation t of  $\bar{o}$  and the translation t' of  $\bar{o}'$ . Note that both t and t' are programs at the conceptual schema level, which facilitates their comparison.

After detecting which influences exist, the next step would be to verify if some of the detected influences are undesirable. But this step involves an independent description of the set of allowed influences. Two bit matrices, M and N, can be used for this purpose, where M has as many rows as there are built-in external schema operations and as many columns as there are external schemas and  $M_{ij} = 1$  indicates that the  $i^{\text{th}}$  built-in operation can influence the  $j^{\text{th}}$  schema; N has as many rows and columns as there are built-in operations and  $N_{ij} = 1$  indicates that the  $i^{\text{th}}$  operation can influence the  $j^{\text{th}}$  operation.

The above discussion then leads to our last property.

**P13. influence consistency:** no undesirable influences occur.

Naturally, if the database description in question does not satisfy Property P13, some external schemas or even the conceptual schema would have to be redefined and the design process iterated.

## 2.6 - Summary

Figure 2.1 lists all properties defined in this section. Not all these properties are compulsory and, in fact, some of them make sense only if built-in operations are used, whereas others reflect the design strategy adopted. Specifically, properties P1 and P3 must be satisfied by all schemas; if the schema definition includes built-in operations, then properties P7, P10, P11 and P13 must also be satisfied. (Note that if the translation of each external built-in operation uses only the conceptual schema built-in operations to modify the database, then P7 implies P11). Independently of the use of built-in operations, property P12 is required if the conceptual schema completely describes the enterprise, but it does not apply if the strategy of introducing certain constraints only through the external schemata is adopted. Properties P2, P8 and P9 are all desirable, but their violation does not create invalid situations from the point of view of the applications users. Property P6 is also desirable, especially when conceptual schemas are considered. Property P4 is compulsory in so far as the database design includes update specifications. Finally, we observe that property P5 can be enforced only if all transitions between valid states are valid, that is, if property P3 is vacuously satisfied due to the absence of any dynamic consistency criteria.

The list of properties now follows.

- P1. consistency: the consistency criteria must not be contradictory.
- P2. logical independence: no consistency criterion is implied by the others.
- P3. dynamic consistency: no state transition (resulting from a sequence of one or more updates) violates any dynamic consistency criteria.
- P4. adequacy of the Schema DML: every update specification must be achievable by some valid update (written in the DML chosen).
- P5. completeness of the Schema DML: given any pair (I, J) of consistent database states, there is a valid update that maps I into J.
- P6. reachability: given any consistent database state I, there is a valid update that maps the initial state into I.
- P7. consistency preservation: each built-in operation must be valid.
- P8. operation independence: no built-in operation can be eliminated without affecting the set of all possible database state transitions.
- P9. operation applicability: for each built-in operation, there is at least one consistent state where the operation can be successfully applied to modify the database.
- P10. translation correctness: the translation  $t$  of an external operation  $\bar{o}$  is correct in the sense that the result of  $\bar{o}$  coincides with the external state constructed from the result of  $t$ .
- P11. conceptual consistency preservation: the translation of each external operation maps the set of consistent database states into itself.
- P12. logical dependence: any external state constructed from a consistent database state is also consistent (with respect to the external consistency criteria).
- P13. influence consistency: no undesirable influences occur.

Figure 2.1  
Schema Properties

### 3. SCHEMAS WITHOUT BUILT-IN OPERATIONS

In the rest of the paper, we formalize the properties listed in Section 2, but following an order dictated by the formalism used. We begin by treating in this section all properties that can be formalized within first-order logic.

#### 3.1 - Special Many-Sorted Languages

Concepts pertaining to the relational model have been formalized in first-order logic by treating relation names as predicate symbols [11, 25]. However, this approach is not adequate when we have to consider functions from relations into relations [10], or when we have to quantify on variables ranging over relations, which is the case in this paper (see Section 4.2). Hence, we are forced to adopt a certain class of many-sorted first-order languages that emulates second-order languages [17 pp. 281, 10] to formalize the relational model.

We say that  $L$  is a special many-sorted (first-order) language iff  $L$  is a many-sorted first-order language with sorts: the individual sort, abbreviated ind, with lowercase letters as variables and, for each  $n > 0$ , the  $n$ -place predicate sort, abbreviated  $n$ -pred, with uppercase letters as variables (superscripted with  $n$  if necessary). We intend the  $n$ -pred domain to be a set of  $n$ -ary relations over individuals.

$L$  must also include the following special parameters, listed with their intended interpretations:

- (1) the equality = of sort (ind, ind) and, for each  $n > 0$ , the  $n$ -pred equality = <sub>$n$</sub>  of sort ( $n$ -pred,  $n$ -pred);
- (2) for each  $n > 0$ , the membership  $e^n$  of sort ( $n$ -pred, ind, ..., ind). The intended interpretation of  $e^n(x^n, x_1, \dots, x_n)$  is that the tuple denoted by  $(x_1, \dots, x_n)$  is

in the  $n$ -ary relation denoted by  $X^n$ ; hence, whenever possible, we abbreviate  $e^n(x^n, x_1, \dots, x_n)$  as  $X^n(x_1, \dots, x_n)$ .

A general structure  $A$  of  $L$  is any structure of  $L$  such that:

- (i)  $A$  has intended domains and assigns to the special parameters their intended interpretations;
- (ii) Let  $P$  be a wff of  $L$  and let the free variables of  $P$  be classified into two disjoint lists  $\bar{x} = (x_1, \dots, x_m)$  and  $\bar{y} = (y_1, \dots, y_n)$ ,  $m \geq 0$ ,  $n \geq 0$ , such that  $x_j$  has sort  $s_j \in \{\text{ind}\} \cup \{\text{i-pred}/i \in \mathbb{N}\}$  and  $y_k$  has sort ind,  $1 \leq j \leq m$ ,  $1 \leq k \leq n$ . Then, the following closed comprehension sentence is true in  $A$ :

$$\forall x_1 \dots \forall x_m \exists \bar{y} \forall y_1 \dots \forall y_n (X^n(y_1, \dots, y_n) \equiv P[\bar{x}, \bar{y}])$$

It is through the concept of general structure that we can truly say that  $L$  emulates a second-order language. This point is further discussed in [17 pp 284, 10]. We write  $\models_A P$  to indicate that  $P$  is valid in a general structure  $A$  of  $L$  and  $P \models P$

to indicate that  $P$  is valid in any general structure (not necessarily in any structure) that satisfies all wffs in  $P$ . If  $t$  is a term of  $L$ ,  $\Lambda(t)$  indicates the element associated with  $t$  by  $A$ . Finally,  $[s/s]A$  indicates the structure  $B$  of  $L$  differing from  $A$  only on the value of symbol  $s$ , which is  $s$  in  $B$ .

The second condition on general structures permits us to add new function symbols to  $L$  by definition. Let  $P[\bar{x}, \bar{y}]$  be a wff of  $L$  with free variables  $\bar{x}, \bar{y}$  (we follow the conventions adopted in the description of general structures). Then, a function symbol  $f$  of sort  $(s_1, \dots, s_m, n\text{-pred})$  can be added to  $L$  by definition with defining axiom

$$(3) \quad f(\bar{x}) = z^n \equiv \forall \bar{y} (z^n(\bar{y}) \equiv P[\bar{x}, \bar{y}])$$

Note that, by definition of general structure, the uniqueness and existence conditions [40, p.59] for  $f$  are satisfied.

By analogy with set theory, we write  $f(\bar{x})$  as  $\{y/P[\bar{x}, \bar{y}]\}$ . In each general structure,  $f$  denotes a function mapping a tuple of elements  $\bar{a} = (a_1, \dots, a_m)$ ,  $a_j$  from the domain of sort  $s_j$ , into the  $n$ -ary relation defined by  $P[\bar{a}/\bar{x}, \bar{y}]$ . These function symbols then create a rich set of  $n$ -pred terms,  $n > 0$ .

This concludes the definition of special many-sorted languages.

#### 3.2 - Conceptual and External Schemas

Before defining what we mean by a schema, it is worth noting that we classify all symbols of a schema into two sets. The first set contains all symbols, such as ' $\leq$ ', whose intended interpretation is fixed. The second set includes all symbols whose meaning varies over time, which will be the relation names in the case of the relational model. Their meaning at a given point in time  $\tau$  comprises what is called the database state at  $\tau$  (however, for simplicity, our definition of database state also includes the meaning of all other symbols).

DEFINITION 3.1:

- (a) A pair  $\sigma = (L, P)$  is a relational schema iff
  - (i)  $L$  is a special many-sorted language with a distinguished set of constant  $\rho = \{r_1, \dots, r_n\}$ ,  $r_i$  of the  $k_i$ -pred sort ( $1 \leq i \leq n$ ), the relation names of  $L$ ;
  - (ii)  $P$  is a set of wffs of  $L$ , the consistency criteria of  $\sigma$ .
- (b) A database state of  $\sigma$  is a general structure of  $L$ ;

- (c) A consistent database state of  $\sigma$  is a database state  $A$  of  $\sigma$  such that  $\models_A P$ , for  $P \in \mathcal{P}$ .
- (d) A database universe  $U$  of  $\sigma$  is a set of database states such that: (i) all database states differ only on the values of the relation names; (ii) for every  $A \in U$ , for every  $n$ -ary relation name  $r$  and every  $n$ -ary relation  $R$  (of the common  $n$ -pred domain), there is  $B \in U$  such that  $B = [R/r]A$ .  $\square$

Thus, we view a schema  $\sigma = (L, \mathcal{P})$  as a first-order theory and a consistent database state of  $\sigma$  as a model of  $\sigma$ . The notion that the meaning of all symbols, except the relation names, is fixed is embodied in the definition of database universe.

We can now readily formalize the basic schema properties listed in Section 2. Let  $\sigma = (L, \mathcal{P})$  be a schema.

- P1. consistency: there is at least one consistent database state of  $\sigma$  or, equivalently,  $\mathcal{P} \not\models \text{False}$ ;
- P2. logical independence:  $\mathcal{P} - \{P\} \not\models P$ , for each  $P \in \mathcal{P}$ .

note: the formalization of Property P3 is deferred to Section 5.

An external schema  $\pi$  of a schema  $\sigma$  is just another schema whose language is basically the same as  $\sigma$ , but which may have its own relation names. As a convenience, we also require that no relation name of  $\sigma$  is used in  $\pi$ . In addition,  $\pi$  contains a function defining its relation names in terms of those of  $\sigma$ .

#### DEFINITION 3.2:

Let  $\sigma = (L, \mathcal{P})$  be a schema.

A triple  $\pi = (M, Q, J)$  is an external schema of  $\sigma$  iff

- (i)  $\pi_0 = (M, Q)$  is a schema such that all symbols of  $M$ , except the relation names, are also symbols of  $L$  and no relation name of  $L$  is a symbol of  $M$ .
- (ii)  $J$  is a function assigning to each  $n$ -ary relation name  $r$  of  $M$  a term  $r^J$  of  $L$  of the same sort as  $r$ .  $\square$

We can extend  $J$  to an interpretation of  $M$  into  $\sigma$  [17, pp 156] as follows:  $J$  considers the domains of  $M$  and  $L$  to be identical;  $J$  is the identity on each symbol of  $M$ , except the relation names; and  $J$  assigns to each relation name  $r$  of  $M$  the wff  $y = r^J$  (this is just to conform with the definition of interpretation).

The function  $J$  can be used for two different purposes. First, given a wff  $P$  of  $M$ , we can construct a wff  $P^J$  of  $L$  by replacing each relation name  $r$  of  $M$  by its definition  $r^J$ , as explained in [17, pp 160].

Second, given a general structure  $A$  of  $L$ , we can construct a structure  $A^J$  for  $M$  as follows. The domains of  $A^J$  are those of  $A$ ; for each relation name  $r$  of  $M$ ,  $A^J(r)$  is the relation  $A(r^J)$ ;  $A^J$  agrees with  $A$  on every other symbol.  $A^J$  is called the structure of  $M$  induced from  $A$  by  $J$ .

We can relate the two constructions by the following lemma.

LEMMA 3.1: For any structure  $A$  of  $L$  and any wff  $P$  of  $M$ ,  $\models_{A^J} P$  iff  $\models_A P^J$ .  $\square$

Lemma 3.1 has several interesting consequences. We begin by showing that Lemma 3.1 implies that the induced structure of  $M$  is indeed a general structure of  $M$ .

COROLLARY 3.1: If  $A$  is a general structure of  $L$ , then  $A^J$  is a general structure of  $M$ .

#### Proof

Let  $A$  be a general structure of  $L$ . Since  $A$  has the intended domains and assigns to the special parameters their intended interpretations, so does  $A^J$ . Let  $P$  be a closed comprehension sentence of  $M$ . Then,  $P^J$  is a closed comprehension sentence of  $L$  and, hence, true in  $A$ . By Lemma 3.1,  $P$  is then true in  $A^J$ . Hence,  $A^J$  also satisfies the second condition of general structures.  $\square$

We now use Lemma 3.1 to formalize Property P12. Let  $\pi = (M, Q, J)$  be an external schema of  $\sigma = (L, \mathcal{P})$ .  $\pi$  and  $\sigma$  satisfy Property P12 iff, for any model  $A$  of  $\mathcal{P}$ , for any  $Q \in \mathcal{Q}$ ,  $A^J$  satisfies  $Q$ . But, by Lemma 3.1  $A^J$  satisfies  $Q$  iff  $A$  satisfies  $Q^J$ . Hence, Property P12 is equivalent to saying that  $Q^J$  is a logical consequence of  $\mathcal{P}$ , which is a much more convenient characterization of Property P12.

P12. Logical dependence:  $\mathcal{P} \models Q^J$ , for each  $Q \in \mathcal{Q}$ .

We now define what we mean by active schema. Given a conceptual schema  $\sigma = (L, \mathcal{P})$  with a set of external schemas  $\pi_i = (M_i, Q_i, J_i)$ ,  $i = 1, \dots, n$ , we say that  $\bar{\sigma} = (L, \bar{\mathcal{P}})$  is the corresponding active schema iff  $\bar{\mathcal{P}} = \mathcal{P} \cup \bigcup_{i=1}^n \{Q_i^J / Q \in \mathcal{Q}_i\}$ . As already discussed,

this concept is important for the second design strategy mentioned in Section 2 since  $\bar{\sigma}$  incorporates all consistency criteria defined for the application, all expressed in the language of the conceptual schema. Using the formalization given in this section for Property P12, it is now obvious that  $\bar{\sigma}$  and  $\pi_i$ ,  $i = 1, \dots, n$ , satisfy Property P12.

This concludes the list of properties that we formalize using first-order logic.

#### 4. SCHEMAS WITH BUILT-IN OPERATIONS

In this section, we discuss schemas with built-in operations. The formalism we use is a variant of Dynamic Logic (DL), quite similar to the one described in [10, 11]. We first briefly describe the DL variant and then address schema properties.

##### 4.1 - Regular Many-Sorted Dynamic Logic

Before defining the DL variant, we need a new concept. Let  $L$  be a special many-sorted language with a set of distinguished constants, henceforth called program variables. A universe  $U$  for  $L$  is a set of general structures of  $L$  satisfying two conditions: (i) any two structures in  $U$  differ only on the values of the program variables; (ii) for any  $A \in U$ , any program variable  $x$  and any element  $e$  of the appropriate domain, there is  $B \in U$  such that  $B = [e/x]A$ . These conditions guarantee that, for example, if  $x$  is assigned  $e$  as value, the resulting structure is in  $U$ . That is, the universe is closed under assignment, so to speak.

The programming language the DL variant uses is the set of regular programs over  $L$ ,  $RP[L]$ , defined inductively as follows:

##### syntax

- (1) for any program variable  $x$  of  $L$  and any term  $t$  of  $L$  such that  $x$  and  $t$  are of the same sort,  $x := t$  is in  $RP[L]$  and is called an assignment;
- (2) for any wff  $P$  of  $L$ ,  $P?$  is in  $RP[L]$  and is called a test;
- (3) for any  $a, b \in RP[L]$ ,  $a \cup b$ ,  $a ; b$  and  $a^*$  are also in  $RP[L]$  and are called the union of  $a$  and  $b$ , the composition of  $a$  and  $b$  and the iteration of  $a$ , respectively.

semantics: for a fixed universe  $U$  of  $L$ , the meaning of programs in  $RP[L]$  is given by a function  $m$  assigning to each  $a \in RP[L]$  a binary relation  $m(a) \subseteq U^2$  as follows:

- (4)  $m(x:=t) = \{(A,B) \in U^2 / B = [A(t)/x]A\}$   
 (5)  $m(P?) = \{(A,A) \in U^2 / \models_A P\}$   
 (6)  $m(a \cup b) = m(a) \cup m(b)$  (union of both binary relations)  
 (7)  $m(a; b) = m(a) \circ m(b)$  (composition of both binary relations)  
 (8)  $m(a^*) = (m(a))^*$  (reflexive and transitive closure of  $m(a)$ )

The language  $\mathcal{DL}$  of the DL variant (based on  $L$ ) is defined as follows:

syntax: the syntax of  $\mathcal{DL}$  is the same as that of  $L$ , with one additional formation rule:

- (1) if  $P$  is a wff of  $L$  or  $\mathcal{DL}$  and  $b \in RP[L]$ , then  $\lceil b \rceil P$  is a wff of  $\mathcal{DL}$  (read "box of  $b, P$ ").

semantics: for a fixed universe  $U$  of  $L$ , the notion of validity is extended to  $\lceil b \rceil P$  as follows:

- (2)  $\models_A \lceil b \rceil P$  iff  $\forall B((A,B) \in m(b) \Rightarrow \models_B P)$

In words,  $\lceil b \rceil P$  is valid in  $A$  iff either  $b$  does not halt starting in  $A$  (that is, for no  $B$  in  $U$ ,  $(A,B) \in m(b)$ ) or, for any state  $B$  that can be reached from  $A$  via  $b$ ,  $P$  is valid in  $B$ .

We also introduce by definition  $\langle b \rangle P$  (read "diamond of  $b, P$ ") as  $\neg \lceil b \rceil \neg P$ . Hence, we have

- (3)  $\models_A \langle b \rangle P$  iff  $\exists B((A,B) \in m(b) \wedge \models_B P)$

In words,  $\langle b \rangle P$  is valid in  $A$  iff program  $b$  takes state  $A$  to some state  $B$  where  $P$  is valid. Finally, we define  $\models_U Q$  iff  $\models_A Q$ , for any  $A \in U$ , where  $Q$  is a wff of  $\mathcal{DL}$ .

The language of the DL variant permits us to express three properties of programs that are central to formalize certain schema properties. Let  $U$  be a universe of  $L$ ,  $P$  be a wff of  $L$ ,  $t_1, \dots, t_k$  be terms of  $L$ ,  $a$  and  $b$  be two programs in  $RP[L]$  and  $x_1, \dots, x_k$  be variables of  $L$  not occurring in  $b$ , where  $x_i$  is of the same sort as  $t_i$ ,  $i = 1, \dots, k$ . Then, we say that:

- (1)  $P$  is an invariant of  $b$  iff  $\models_U P \Rightarrow \lceil b \rceil P$   
 (2) the values of  $t_1, \dots, t_k$  are unaffected by  $b$  iff  $\models_U \forall x_1 \dots \forall x_k (Q \Rightarrow \lceil b \rceil Q)$ , where  $Q = \bigwedge_{i=1}^k x_i = t_i$   
 (3) programs  $a$  and  $b$  are equivalent for  $t_1, \dots, t_k$  iff  $\models_U \forall x_1 \dots \forall x_k (\langle a \rangle Q \equiv \langle b \rangle Q)$ , where  $Q = \bigwedge_{i=1}^k x_i = t_i$

#### 4.2 - Conceptual and External Schemas Revised

We are now ready to define schemas with built-in operations.

DEFINITION 4.1:

- (a) A schema with built-in operations is a triple  $\sigma = (L, P, \mathcal{O})$  such that  
 (i)  $(L, P)$  is a schema such that  $L$  contains a distinguished set of constants called program variables, which include the relation names;  
 (ii)  $\mathcal{O}$  is a set of programs in  $RP[L]$

- (b) An external schema with operations of  $\sigma$  is a four-tuple  $\pi = (M, Q, J, N)$  such that  
 (i)  $(M, Q, N)$  is a schema with operations such that all symbols of  $M$ , except the relation names, are also symbols of  $L$  and no relation name of  $L$  is a symbol of  $M$ .  
 (ii)  $J$  is a function assigning to each  $n$ -ary relation name  $r$  of  $\pi$  a term  $r^J$  of  $L$  of the same sort as  $r$  and to each operation  $a \in N$  a program  $a^J$  of  $RP[L]$ .  $\square$

The notions of database state and consistent database state are as in Definition 3.1. However, we adopt the concept of universe defined in Section 4.1, rather than that of Definition 3.1. This follows because not only the relation names, but certain other constants (i.e., the other program variables) may change value from state to state.

We now discuss the role of the set  $\mathcal{O}$  of built-in operations of a schema  $\sigma$ . Recall from Section 2.3 that we considered as the DML of  $\sigma$  the set of regular programs that used only operations in  $\mathcal{O}$  to update the database. In the abstract data type jargon, this strategy is called "encapsulation".

More precisely, we define the set  $RP_{\mathcal{O}}[L]$  of restricted regular programs for a schema  $\sigma$  with a set  $\mathcal{O}$  of built-in operations as follows:

- (1) if  $a \in \mathcal{O}$ , then  $a \in RP_{\mathcal{O}}[L]$ ;  
 (2) if  $b$  is an assignment of  $RP[L]$  such that the left-hand side is not a relation name of  $\sigma$ , then  $b \in RP_{\mathcal{O}}[L]$ .  
 (3) if  $P$  is a wff of  $L$ , then  $P? \in RP_{\mathcal{O}}[L]$ ;  
 (4) if  $a, b \in RP_{\mathcal{O}}[L]$ , then  $a; b$ ,  $a \cup b$  and  $a^*$  are in  $RP_{\mathcal{O}}[L]$ .

The relevance of encapsulation by means of built-in operations lies in that we can prove that, if each built-in operation preserves consistency, then so does each program in  $RP[L]$ . Therefore, users cannot violate the consistency of the database. To prove this result, we start with a basic lemma.

LEMMA 4.1: Let  $\sigma = (L, P, \mathcal{O})$  be a schema with a set  $\mathcal{O} = \{o_1, \dots, o_n\}$  of built-in operations, and  $b \in RP_{\mathcal{O}}[L]$ . Construct the regular program  $p = (o_1 \cup \dots \cup o_n)^*$ . Then, for any  $(A, B) \in m(b)$ , there is  $(A, C) \in m(p)$  such that  $B$  and  $C$  are equal on the relation names of  $\sigma$ .

Proof

Let  $b \in RP_{\mathcal{O}}[L]$  and  $(A, B) \in m(b)$ . Since  $b$  uses only operations in  $\mathcal{O}$  to modify the database, there is a sequence  $q = (o_{i_1}, \dots, o_{i_k})$  of operations in  $\mathcal{O}$ , generated by the execution of  $b$  that takes  $A$  to  $B$ , such that  $q$  takes  $A$  to  $C$  and  $B$  and  $C$  are equal on the relation names of  $\sigma$ . But, by definition of  $p$ ,  $(o_{i_1}, \dots, o_{i_k})$  is also a possible execution of  $p$ . Hence,  $(A, C) \in m(p)$ .  $\square$

COROLLARY 4.2: Let  $\sigma = (L, P, \mathcal{O})$  be a schema and assume that no program variable, other than the database relation names of  $\sigma$ , occurs in wffs of  $P$ . Then, if each operation in  $\mathcal{O}$  preserves consistency, then every program in  $RP_{\mathcal{O}}[L]$  preserves consistency.

Proof

Let  $b \in RP_{\mathcal{O}}[L]$ . Let  $(A, B) \in m(b)$  and assume that  $A$  is consistent. By Lemma 4.1, there is  $C \in U$  such that  $(A, C) \in m(p)$ , where  $p = (o_1 \cup \dots \cup o_n)^*$ , and  $B$  and  $C$  are equal on the relation names of  $\sigma$ . But, by assumption, each  $o_i$ ,  $i = 1, \dots, n$ , preserves consistency. So,  $p$  also preserves consistency. Therefore, since  $A$  is consistent,  $C$  is also consistent. Thus, by the assumption on  $P$ ,  $B$  is also consistent. Hence, we may conclude

that every  $b \in RP_{\sigma}[L]$  preserves consistency.  $\square$

To summarize, in the case of encapsulation through built-in operations, users cannot violate consistency of the database since every possible DML program preserves consistency.

4.3.- Formalization of Properties Related to Built-in Operations.

Within the framework developed here, we can completely formalize all properties related to built-in operations. Properties P7, P9 and P11 follow directly using the concepts introduced at the end of Section 4.1. Let  $\sigma = (L, P, O)$  be a schema and  $U$  be a universe for  $L$ .

P7. consistency preservation: for each  $b \in O$ , for each  $P \in \mathcal{P}$ ,  $P$  is an invariant of  $b$  (for  $U$ ).

P9. operation applicability: for each  $b \in O$ , it is not the case that all relation names of  $\sigma$  are unaffected by  $b$ .

Let  $\sigma$  and  $U$  be as above and  $\pi = (M, Q, J, N)$  be an external schema of  $\sigma$ .

P11. conceptual consistency preservation: for each  $B \in N$ , for each  $P \in \mathcal{P}$ ,  $P$  is an invariant of  $b^J$  (for  $U$ ).

Consider now Property P8, operation independence, which requires that no operation  $o_i$  is superfluous. To formalize Property P8, we recall from Lemma 4.1 that the set of all database transitions that can be performed by finite sequences of operations coincides with the set of database transitions that can be brought about by the regular program  $p = (o_1 u_1 \dots u_n o_n)^*$ . Let  $p_i = (o_1 u_1 \dots u_{i-1} u_{i+1} \dots u_n o_n)^*$ . Then,  $p_i$  captures all database transitions that can be executed without the help of  $o_i$ . Therefore, we can concisely state Property P8 as follows:

P8. operation independence: for any  $i \in \{1, n\}$ ,  $p$  is not equivalent to  $p_i$  for the relation names of  $\sigma$ .

Property P10, translation correctness, is not so easy to formalize, though. So, let us then try to precisely define it first. Let  $\pi = (M, Q, J, N)$  be an external schema of  $\sigma = (L, P, O)$ . Assume that  $r_1, \dots, r_m$  are the relation names of  $\pi$ . Let  $U$  be the universe of  $L$  in question. Then, Property P10 says the following. Let  $A \in U$ . Apply  $b^J$  to  $A$ , obtaining some state  $B \in U$ . Now apply  $b$  to  $A^J$  obtaining  $C$ . Then,  $B^J$  and  $C$  must agree on  $r_1, \dots, r_m$  (informally  $B^J \sim C$ ). Conversely, let  $A \in U$ . Apply  $b$  to  $A^J$ , obtaining some state  $C$  of  $\pi$ . Then, there must be some state  $B$  attainable by  $b^J$  from  $A$  such that  $C$  and  $B^J$  agree on  $r_1, \dots, r_m$ . This can be expressed concisely by the diagram in Figure 4.1.



Figure 4.1

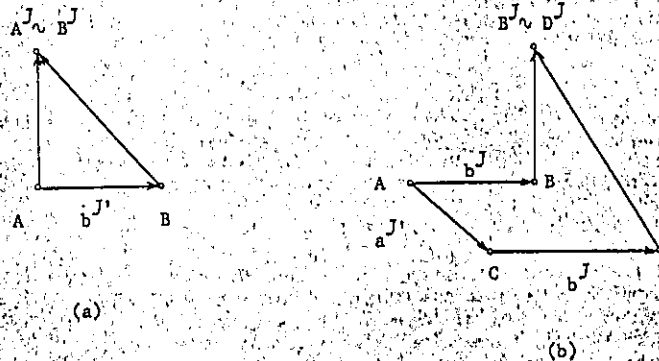
Now we observe that  $J$  can be emulated by a program  $q$  in the set of regular programs  $RP[LUM]$ , if we adopt the universe  $V$  of  $LUM$  constructed from  $U$  by using  $J$  to extend each  $A \in U$  to the relation names of  $\pi$ . Let  $r_1, \dots, r_m$  be the relation names of  $\pi$  and recall from section 4.2 that  $r_i^J$  is a term of the same sort as  $r_i$ . Then, we have:

$$q = r_1 := r_1^J; \dots; r_m := r_m^J$$

Combining all these observations with the notion of program equivalence, P10 can be formalized as follows:

P10. translation correctness:  $b^J$  correctly translates  $b$  iff  $b^J; q$  and  $q; b$  are equivalent for  $r_1, \dots, r_m$ .

To investigate Property P13, influence consistency, we first have to formalize the notion of operation influence. Let  $\pi = (M, Q, J, N)$  and  $\pi' = (M', Q', J', N')$  be two external schemas of  $\sigma = (L, P, O)$ . An operation  $b \in N'$  does not influence any query of  $\pi$  iff, given any two states  $A$  and  $B$  of  $\sigma$  such that  $b^{J'}$  maps  $A$  into  $B$ , then  $A^J$  and  $B^J$  agree on the relation names  $r_1, \dots, r_m$  of  $\pi$ . Using the concepts at the end of Section 4.1, we then say that  $b \in N'$  does not influence any query of  $\pi$  iff  $r_1, \dots, r_m$  are unaffected by  $b^{J'}$ . The diagram of Figure 4.2(a) may help understand this definition.



(Figure 4.2)

Likewise, we say that  $a \in N'$  does not influence  $b \in N$  iff  $a^{J'}; b^J$  and  $b^J; a^{J'}$  are equivalent for  $r_1, \dots, r_m$  (c.f. the diagram in Figure 4.2(b)).

Now the formalization of Property P13 follows:

P13. Influence Consistency: all influences detected (using the definitions above) are allowed.

4.4 - Properties Related to the DML

We now discuss P4, P5 and P6. All these properties are related to the existence of programs satisfying certain restrictions. But this cannot be expressed in (First-Order) DL since we cannot quantify over programs. For example, Property P4 would have to be expressed as follows:

P4. Adequacy of the Schema DML:  $\vdash_U \exists b (\bar{P}AP \Rightarrow [b]PAQ)$  for each update specification  $(P, Q)$ , where  $U$  is the universe in question and  $\bar{P}$  is the conjunction of all consistency criteria.

Note that we used a variable  $b$  ranging over the set of allowed programs. Hence, the wff above is phrased within what we may call Second-Order Dynamic Logic.

However, when we consider a schema  $\sigma$  with a set  $O = \{o_1, \dots, o_n\}$  of built-in operations, we can formalize Properties P5 and P6. We assume that no program variable, other than the relation names of  $\sigma$ , occurs in any consistency criteria of  $\sigma$ . We also need additional notation. If  $P$  is a consistency criterion of  $\sigma$ , let  $P[\bar{x}]$  denote the wff obtained by replacing each relation name  $r_i$  by a variable  $x_i$  of the same sort as  $r_i$ ,  $1 \leq i \leq n$ . Let  $\bar{P}[\bar{x}]$  be the conjunction of the wffs thus formed. Then, for example, the wff  $\exists x \bar{P}[\bar{x}]$  says that there is a consistent database state of  $\sigma$ .

Consider now Property P5, which says that, given any pair  $(A, B)$  of consistent database states, there must be some valid update that maps  $A$  into  $B$ . By Lemma 4.1, this is equivalent to requiring that  $p = (o_1 U \dots U o_n)^*$  takes any consistent database state into any other consistent database state:

P5. Completeness of the Schema DML:

$$\vdash_U \forall \bar{x} \forall \bar{y} (\bar{P}[\bar{x}] \wedge \bar{P}[\bar{y}] \Rightarrow \langle p \rangle \bigwedge_{i=1}^m x_i = r_i = \langle p \rangle \bigwedge_{i=1}^m y_i = r_i)$$

where we assume that no variable in  $\bar{x}$  or  $\bar{y}$  occurs in  $p$ .

Assuming that  $P_0$  characterizes the initial state, Property P6 then reads:

P6. Reachability:

$$\vdash_U \forall \bar{x} (\bar{P}[\bar{x}] \wedge P_0 \Rightarrow \langle p \rangle \bigwedge_{i=1}^m x_i = r_i)$$

where no variable in  $\bar{x}$  occurs in  $p$ .

### 5. SCHEMAS WITH DYNAMIC CONSISTENCY CRITERIA

The schemas considered thus far contained only consistency criteria restricting data values. In this section, we investigate schemas with dynamic consistency criteria imposing restrictions directly on state transitions. For example, suppose that  $\sigma$  is a schema with just one binary relation name EMP. We interpret EMP( $m, s$ ) as saying that the employee whose social security number is  $m$  has salary  $s$ . Then,  $D_1$  and  $D_2$  are dynamic consistency criteria, where  $D_1$  is "an employee's salary should never decrease" and  $D_2$  is "an employee that is fired (i.e., deleted from EMP) can never be readmitted again (i.e., inserted into EMP again)".

We first observe that Dynamic Logic is not quite appropriate to describe dynamic consistency criteria because these criteria impose restrictions on a state transition, no matter what update or sequence of updates performed the transition (by an update we understand any program of the schema DML). For example, the criterion  $D_2$  would be expressed as:

(1) for any two sequences  $p$  and  $q$  of updates,

$$\neg \exists m \exists s \exists s' \exists s'' (EMP(m, s) \wedge \langle p \rangle (\neg EMP(m, s')) \wedge \langle q \rangle EMP(m, s''))$$

However, note that  $p$  and  $q$  act as variables ranging over sequences of updates. Hence,

(1) is not a wff of (First-Order) DL. But in such cases, unlike the case of property P4, we do not really want to refer explicitly to specific sequences of updates; accordingly, we are led to consider some formalism wherein we could rewrite formula (1) above omitting somehow variables  $p$  and  $q$ .

So we now briefly discuss how we can use Modal Logic [13] to express dynamic consistency criteria. Given a special many-sorted language  $L$ , we first define the

language  $ML$  of the Modal Logic based on  $L$  by adding one more formation rule:

(1) If  $P$  is a wff of  $L$  or  $ML$ , then  $\Box P$  is also a wff of  $ML$ .

We also introduce by definition  $\Diamond P$  as  $\neg \Box \neg P$ .

A structure  $I$  for  $ML$  is a pair  $(U, R)$ , where  $U$  is a universe for  $L$  and  $R \subseteq U^2$  ( $R$  is called a universe of state transitions). We extend the notion of validity to  $\Box P$  as follows. Let  $I = (U, R)$  be a structure of  $ML$  and  $A \in U$ :

(2)  $\vdash_A^I \Box P$  iff  $(\forall B \in U)((A, B) \in R \Rightarrow \vdash_B^I P)$

(3)  $\vdash_A^I \Diamond P$  iff  $(\exists B \in U)(\vdash_B^I P)$

Hence, since  $\Diamond P$  is  $\neg \Box \neg P$ , we have

(4)  $\vdash_A^I \Diamond P$  iff  $(\exists B \in U)((A, B) \in R \wedge \vdash_B^I P)$

(5)  $\vdash_A^I \Box P$  iff  $(\forall A \in U)(\vdash_A^I \Diamond P)$

We then define a schema with dynamic consistency criteria as a triple  $\sigma = (L, P, D)$  where  $(L, P)$  is as in Section 3 and  $D$  is a set of wffs of  $ML$ . In our running example,  $D_1$  and  $D_2$  would be defined respectively as:

(1)  $\forall e (EMP(e, s) \Rightarrow \Box (\forall m \forall s' (EMP(m, s) \wedge EMP(m, s') \Rightarrow s \leq s')))$

(2)  $\neg \exists m \exists s \exists s' \exists s'' (EMP(m, s) \wedge \Diamond (\neg EMP(m, s')) \wedge \Diamond EMP(m, s''))$

A structure  $I = (U, R)$  for  $ML$  is valid iff  $I$  satisfies all criteria in  $D$ ;  $I$  is adequate iff  $R$  contains exactly all pairs  $(A, B) \in U^2$  such that there is a (possibly empty) finite sequence of updates that maps  $A$  into  $B$ . A valid structure is then one that conforms with all dynamic consistency criteria, and an adequate structure is one that provides the intended interpretation of these criteria. Let  $\sigma = (L, P, D)$  be a schema and  $I$  be an adequate structure for  $ML$ . Property P3 then reads:

P3. Dynamic Consistency:  $I$  is valid.

Since this new property is almost impossible to guarantee in unrestricted schemas, we are again led to consider schemas with built-in operations. The design goal of built-in operations in this context is then to guarantee that no sequence of programs that modify the database through these operations violates the dynamic consistency criteria. We show in the rest of this section that this goal is somewhat simpler to achieve than it seems.

Let  $\sigma = (L, P, D, O)$  be a schema where  $L$  and  $P$  are as usual,  $D$  is a set of dynamic consistency criteria and  $O$  is a set of built-in operations. Recall from Section 4 that we took as the schema DML the set  $RP_\sigma[L]$  of regular programs that modify the database only through operations in  $O$ . Let  $I = (U, R)$  be an adequate structure of  $L$ . Then  $R$  consists of all pairs  $(A, B) \in U^2$  such that there is a (possibly empty) finite sequence  $p = (p_1, \dots, p_n)$  of programs in  $RP_\sigma[L]$  that takes  $A$  into  $B$ . We observe that the design of the built-in operations influences the validity of  $I$  only indirectly through the programs in  $RP_\sigma[L]$ .

We now show that there is a structure  $I'$  whose definition is much more intimately related to  $O$ . Moreover, although  $I'$  is not an adequate structure,  $I'$  is in a precise sense equivalent to  $I$ . We assume in the rest of this section that no program variable, except the relation names of  $\sigma$ , occurs in any wff of  $D$ . Hence, the validity of each  $D \in D$  depends only on symbols whose meaning is fixed and on the value of the relation names.

Define  $I' = (U, R')$  where  $R' = (m(o_1)U \dots U m(o_n))^*$  and  $O = \{o_1, \dots, o_n\}$ . Note that  $R'$  has a much simpler description than  $R$ . We claim that:

**THEOREM 5.1:** For any  $D \in \mathcal{D}$ , for any  $A \in U$ ,  $\models_A^I D$  iff  $\models_A^{I'} D$ .

**Proof**

It suffices to prove that

- (1)  $\models_A^I \Box P$  iff  $\models_A^{I'} \Box P$   
 (2)  $\models_A^I \neg \Box P$  iff  $\models_A^{I'} \neg \Box P$

since, using (1) and (2), we can prove by induction on the structure of  $D \in \mathcal{D}$  that

$\models_A^I D$  iff  $\models_A^{I'} D$ . We first prove that

- (3)  $R' \subseteq R$

Let  $p = (o_1 U \dots U o_n)^*$ . Then,  $m(p) = R'$ . But  $p \in RP_0[L]$ . Hence  $R' \subseteq R$ , by definition of  $R$ .

Now, by adapting the proof of Lemma 4.1, we can prove that:

- (4)  $(\forall (A, B) \in R)(\exists (A, C) \in R') (B \vee C)$

where  $B \vee C$  indicates that  $B$  and  $C$  agree on the relation names of  $\sigma$ .

Now, it is easy to prove that (3) implies that

- (5)  $\models_A^I \Box P$  implies  $\models_A^{I'} \Box P$   
 (6)  $\models_A^I \neg \Box P$  implies  $\models_A^{I'} \neg \Box P$

and that (4), together with our assumption about wffs in  $\mathcal{D}$ , implies the converse of (5) and the converse of (6).  $\square$

Theorem 5.1 has two important consequences. First, consider the task of designing built-in operations so that no sequence of programs that use them to modify the database violates the dynamic criteria. By Theorem 5.1, it suffices to design the operations so that all dynamic criteria are valid in  $I' = (U, R')$ , where  $R' = (m(o_1)U \dots U m(o_n))^*$  and  $o_1, \dots, o_n$  are the built-in operations. But since  $R'$  is defined in terms of sequences of built-in operations (and not sequences of programs) the second task is much simpler than the first. This was the reason for introducing  $I'$ .

Second, by the proof of Theorem 5.1, we have  $R' = m(p)$ , where  $p = (o_1 U \dots U o_n)^*$  is a regular program over  $L$ . Therefore, given  $I' = (U, R')$ ,  $\models_A^{I'} \Box P$  iff  $\models_U^{I'} [p]P$ , and  $\models_A^I \neg \Box P$  iff  $\models_U \neg [p]P$ . But this implies that, for any wff  $Q'$  of  $ML$ , we can find a wff  $Q$  of  $DL$  such that  $\models_A^{I'} Q'$  iff  $\models_U Q$ . As a consequence, with the help of  $p = (o_1 U \dots U o_n)^*$ , we can state all dynamic consistency criteria of a schema with built-in updates  $o_1, \dots, o_n$  as wffs of  $DL$ . (The underlying assumption that no program variables other than the relation names are used in dynamic consistency criteria should not be forgotten).

To summarize, dynamic consistency criteria can best be discussed within the framework of Modal Logic. However, when schemas with built-in operations are considered we have the option to use Dynamic Logic to express these constraints. Naturally, auxiliary data structures summarizing information about the past states of the databases (e.g. names of former employees) would probably be needed to construct the built-in operations.

Finally, we observe that there are sentences about databases that explicitly refer to time. Examples are "two years after admission, any graduate student must have

completed all credits", "an employee can only have his salary raised six months after being hired", and "at a specified date, a fine is added to the amount in debt, if the debt is not paid". Such sentences arise quite naturally when one considers "active" databases [31] (a concept we have not covered), which are capable of initiating action rather than merely acting in response to user requests. Although these sentences are somewhat similar to dynamic consistency criteria, they are best discussed using formalisms such as those described in [12, 37].

## 6. CONCLUSIONS

Taken together, the properties defined constitute our notion of correctness of data base design, with respect to conceptual and external schemas.

We have found that, although there has been considerable research on the subject, it has been rather fragmentary. Moreover, very often too much effort is spent on problems that are not relevant to properties that we consider fundamental. For example, most research on dependency theory would help only check if some dependency is logically independent from the others (Property P2). But very few papers in this area address the other (certainly more important) properties we listed.

By taking the point of view of one who is developing a database design methodology, we strived for breadth and comprehensiveness. In some cases the formal language used to characterize a property was part of a formal system that also provided tools to verify if a given database design satisfied the property. In other situations, we relied on the expressive power of a formal language only to convey our ideas more precisely.

We used first-order logic to formalize the basic design concepts and properties, and moved to dynamic logic when considering built-in operations. Strictly speaking, this was not an essential move, since we can describe properties involving built-in operations in first-order logic [43]. However, we believe that dynamic logic, which gives special attention to programs, provides a much more convenient notation to describe these properties. Similar remarks apply to modal logic. In fact, we showed at the end of Section 5 that dynamic constraints can be formulated within dynamic logic when built-in operations are used. Again, this observation does not render modal logic superfluous because the latter provides a much more convenient notation to describe these constraints.

At several points the convenience of design strategies involving built-in operations has been demonstrated. In particular, it is much simpler to characterize how the various schemas interact if, at both the conceptual and external levels, only built-in operations are considered and, in addition, the external operations are defined using the conceptual ones.

The concept of active schema defined from the conceptual and external schemas also proved very useful. Consider a design strategy where the conceptual schema contains only stable consistency criteria, whereas the external schemas contain criteria that describe less permanent restrictions. Such strategy suits better an evolving environment, and the single collective description, that currently holds, of the whole enterprise is provided by the active schema.

Finally, we observe that, by listing basic schema design properties, we tried to map the area so that future research may proceed in an orderly fashion, attacking each property in turn. For example, tools may be developed to help synthesize built-in operations from non-procedural specifications that take into account the consistency criteria. The derivation of translations for the built-in operations of external schemas may also be partly automated, perhaps requiring the intervention of some administrator to resolve translation ambiguities.

## ACKNOWLEDGEMENT

This research was supported in part by FINEP and by CNPq grants 402090/80 and 102451/78.

## REFERENCES

- [1] A.V.Aho, C.Beerl, and J.D.Ullman. "The Theory of Joins in Relational Databases". ACM Trans. on Database Systems 4.3 (Sept. 1979), 297-314.
- [2] "Study Group on Data Base Management Systems: Interim Report". FDT 7.2, ACM (1975).
- [3] F. Bancilhon. "Supporting View Updates in Relational Data Bases". In Data Base Architecture (G. Bracchi and G.M. Nijssen, eds.), North-Holland (1979).
- [4] C.P.Beerl, P.A.Bernstein, and N.Goodman. "A Sophisticate's Introduction to Database Normalization Theory". Proc. 4th Int. Conf. on Very Large Data Bases (1978), 113-124.
- [5] C.Beerl, R.Fagin, and J.H.Howard. "A Complete Axiomatization for Functional and Multivalued Dependencies". Proc. ACM SIGMOD Conf. (Aug. 1977), 47-61.
- [6] P.A.Bernstein and N.Goodman. "What does Boyce-Codd Normal Form do?". Proc. 6th Int. Conf. on Very Large DataBases (1980).
- [7] M.L.Brodie. "On Modelling Behavioural Semantics of Databases". Proc.7th Int.Conf. on Very Large Data Bases (1981), 32-42.
- [8] F.Bancilhon and N.Spyratos. "Data Base Mappings: Part I - Theory, Part II - Application". R.R.62 and R.R.63. INRIA (April 1981).
- [9] M.A.Casanova. "A Theory of Data Dependencies over Relational Expressions". Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles (March 1982).
- [10] M.A.Casanova and P.A.Bernstein. "A Formal System for Reasoning about Programs Accessing a Relational Database". ACM Trans. on Programming Languages 2.3 (July 1980), 386-414.
- [11] M.A.Casanova and P.A.Bernstein. "The Logic of a Relational Data Manipulation Language". Proc. 5th ACM Symp. Principles of Programming Languages (Jan. 1979) 101-109.
- [12] J.M.V.Castilho, M.A.Casanova, A.L.Furtado. "A Temporal Framework for Database Specifications". TR DB038201, Dept. Informática, Pontificia Univ. Católica do Rio de Janeiro (March 1982).
- [13] B.F.Chellas. "Modal Logic: An Introduction". Cambridge Univ. Press (1980).
- [14] E.F.Codd. "A Relational Model for Large Shared Data Banks". Commun. ACM 13.6 (June 1970), 377-387.
- [15] C.J.Date. "An Introduction to Database Systems". Addison-Wesley (1981).
- [16] U.Dayal and P.A.Bernstein. "On the Updatability of Relational Views". Proc. 4th Int. Conf. on Very Large Data Bases (1978), 368-377.
- [17] H.B.Enderton. "A Mathematical Introduction to Logic". Academic Press (1972)
- [18] R.Fagin. "Multivalued Dependencies and a New Normal Form for Relational Databases". ACM Trans. on Database System 2.3 (Sept. 1977), 262-278.
- [19] R.Fagin. "Horn Clauses and Database Dependencies". Proc.ACM Symp. on Theory of Computation (1980), 123-134.
- [20] A.L.Furtado, C.S. dos Santos and J.M.V. de Castilho. "Dynamic Modelling of a Simple Existence Constraint". Information Systems 6.1 (1981), 73-80.
- [21] A.L.Furtado, K.C.Sevcik, and C.S. dos Santos. "Permitting Updates through Views of Data Bases". Information Systems 4.4 (1979), 269-283.
- [22] H.Gallaire. "Impacts of Logic on Data Bases". Proc. 7th Int. Conf. on Very Large Data Bases (1981), 248-259.
- [23] J.Grant and B.E.Jacobs. "On Generalized Dependencies" (to appear).
- [24] G.Gardarin and M.Melkanoff. "Proving Consistency of Database Transactions". Proc. 5th Int. Conf. on Very Large Data Bases (1979), 291-298.
- [25] H.Gallaire, J.Minker and J.M.Nicolas (eds.). "Advances in Data Base Theory". Plenum Press (1981).
- [26] D.Harel. "First-order Dynamic Logic". In Lecture Notes in Computer Science, 68. Springer-Verlag (1979).
- [27] W.Kent. "Splitting the Conceptual Schema". Proc. 6th Int. Conf. on Very Large Data Bases (1980), 10-14.
- [28] A.Klug. "Entity-Relationship Views over Uninterpreted Enterprise Schemas". Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design (1979), 52-72.
- [29] B.H.Liskov, A.Snyder, R.Atkinson, and C.Schaffert. "Abstraction Mechanisms in CLU". Commun. ACM 20-8 (August 1977), 564-576.
- [30] D.Maier, A.Mendelzon, and Y.Sagiv. "Testing Implications of Data Dependencies". ACM Trans. on Database Systems 4.4 (Dec. 1979), 455-469.
- [31] C.A.Montgomery and E.H.Ruspini. "The Active Information System: a Data-Driven System for the Analysis of Imprecise Data". Proc. 7th Int. Conf. on Very Large Data Bases (1981), 376-384.
- [32] J.M.Nicolas. "First Order Logic Formalization for Functional, Multivalued, and Mutual Dependencies". Proc. ACM-SIGMOD (1978), 40-46.
- [33] E.J.Neuhold and T.Olnhoff. "Building Data Base Management Systems through Formal Specifications". Proc. Formalization of Programming Concepts in Computer Science. Lecture Notes in Computer Science 107. Springer-Verlag (1981).
- [34] P.Paolini. "An Alternative Structure for Data Base Management Systems". Proc. 4th Int. Conf. on Very Large Data Bases (1978), 243-254.
- [35] P.Paolini and G.Pelagatti. "Formal Definition of Mappings in a Data Base". Proc. ACM-SIGMOD (1977), 40-46.
- [36] V.R.Pratt. "Semantical Considerations on Floyd-Hoare Logic". Proc. 17th IEEE Symp. Foundations of Computer Science (Oct. 1976), 109-120.
- [37] N.Rescher, A.Urquhart. "Temporal Logic". Springer-Verlag (1971).