Formal semantics for BRM with examples

Eindhoven University of Technology

Department of Mathematics and Computing Science

Formal Semantics for BRM with examples

by

A.T.M. Aerts D. de Reus

Computing Science Note 91/24
Eindhoven, November 1991

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Formal Semantics for BRM

with examples

A.T.M. Aerts and D. de Reus
Department of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven. the Netherlands

email: wsinatma@win.tue.nl, dickr@win.tue.nl

Abstract

The Binary Relationship Model (BRM) is a widely used meta-model in conceptual modeling because it is part of a methodology for information analysis, NIAM, and because tools exist to map BRM conceptual models onto database definitions. The growing need to define semantics in conceptual models demands several extensions to BRM, among which is the definition of a constraint language. Because BRM has no formal semantic definition, the first step is to derive semantics for BRM schemas. In this paper, we give a formal definition of the semantics of a BRM schema and show the equivalence of BRM schemas to FDM schemas by giving a general mapping algorithm between these two schemas. Because the Functional Data Model has a well-defined semantics in terms of sets and functions, as well as a data language for expressing queries and constraints, the algorithm opens the way to well-defined extensions on the semantics of BRM. The extension of the results to N-ary relationship models is discussed.

1 Introduction

Conceptual modeling is an important step in the development of information systems, because it results in a model which is the basis for all further steps. Several methods for conceptualizing exist, most of which are incorporated in information system development methodologies. A conceptual data model should give a specification of the state space of a part of the real world, referred to as the *object system*, in terms of entities and their relations, which can be used by designers to develop an information system. It should therefore cover the structural, semantic, and dynamic aspects of the object system. A meta-model gives rules for building a model, that is, which kinds of components are to be used to describe the object system, which kinds of structures are to be defined, and how semantic information is to be included in conceptual models.

In the Binary Relationship Model (BRM) [Sen75, Nij77] the basic components are called object types, and the structure is given by binary relationships and sub-type hierarchies. The functional data model FDM [ABvH90] also uses object types as basic components, and functions and subtyping as structuring facilities.

The increasing importance of semantic modeling in software engineering and database technology makes clear the lack of a formal semantics for BRM. This is most clearly signalled by the absence of a data language for formulating constraints which cannot be expressed graphically, and queries. For FDM, a formal semantics and a data language have been defined [ABvH90].

Another aspect of BRM is its generality: it conceptualizes in terms of classes and binary relations between classes. In practice, however, relations between classes often are restricted to functions. This leads to constraints on the BRM primitive concept 'relation', in order to express the functional character of relations. A functional data model needs no constraints to express functions because they are primitives in this model. However, to represent pure binary relations in FDM additional entities must be introduced.

Thus, from a software engineering point of view, BRM is easier to use. From a formal point of view, FDM is best suited for a semantic definition of concepts. By mapping a BRM schema to an FDM schema, one can provide formal semantics for the BRM schema, preserving BRM's user friendliness. Reversely, by converting an FDM schema to a BRM schema, one can make use of the extensive CASE tools available for BRM. In this paper, we will pursue the first option. The relationship between FDM and BRM has recently become interesting also for another reason. In the ESPRIT-project PROOFS [1] the use of formal methods for the development of distributed industrial applications is studied. In this project the aim is not to define a new integrated formalism that covers all the aspects, but rather to develop linkage of several existing methods and supporting tools. One of these tools is ExSpect, a tool for writing executable specifications [vHSV89], the data model of which is based on a type system using, among others, sets and functions as basic types, the mathematical concepts underlying FDM. On the data modeling side, the

[1] Esprit project EP5342, the acronym stands for: Promotion of Formal Methods in European Software Industry.

PROOFS project considers BRM. The link between the two is provided by this paper.

The structural aspects of the Binary Relationship Model are discussed briefly in section 2. A formal definition is given in Appendix A. An example is introduced which will serve as the running example throughout the paper. In section 3, we briefly discuss the Functional Data Model, for which a formal definition has been given in [ABvH90], together with a formal, denotational semantics in terms of sets and functions. The algorithm for mapping a BRM instance onto an FDM instance is presented in section 4. We distinguish between two phases: a straightforward mapping phase, and a rewriting phase in which redundancy resulting from the transformation of the preceding phase is eliminated. In section 5, we summarise the results and discuss some open questions.

2 Binary Relationship Model

The Binary Relationship Model (BRM) was introduced in [Sen75]. Although several papers have been written on the model (for example, [Mar87, Nij77, NH89]), none of them deals with a formal basis for it, each giving a slightly different informal definition. A formal definition for BRM is given in appendix A. In the sequel we will informally comment on the aspects of BRM we consider there.

concepts A conceptual model of an object system given in terms of BRM consists of *object types*, representing entities, and *fact types*, representing relations between entities. In the NIAM methodology ([NH89, Win87, Hab88, AZ90]), object types are derived from the nouns in a textual description of the object system, and fact types are derived from the verbs. An object type is given a unique name and describes a domain [2] of objects. A distinction is made in BRM between object types representing 'lexical entities', or printable objects, such as '*1305*' and '*John*', and object types representing 'non-lexical entities', such as a person or an examination. A domain of lexical objects (for example, {'*1*', ..., '*100*'} or {'*John*', '*Mary*', '*Paul*'}) is called a lexical object type (LOT) a domain of non-lexical objects is called a non-lexical object type (NOLOT), e.g., '*person*' may be the name of a NOLOT representing a domain of persons.[3]

Relations between objects are called *facts*, relationships between object types are represented by *fact types* which are restricted to binary fact types in BRM. A binary fact type may be regarded as a Cartesian product of two object types.

Sometimes an object type does not give enough information for certain subdomains, and subdivision of its domain into subdomains is needed. This *specialization* is represented in BRM by *subtype links*. A subtype link between two object types denotes an *is_a* relation between their domains. Each object belonging to the sub object type also belongs to the

[2]The term *domain* is used here to denote a base set. Each subset is a valid choice for an instance of the object type.

[3]There exist more suitable names: LOTs denote *label types*, and NOLOTs denote *entity types*. However, NIAM literature still holds on to the original names 'LOT' and 'NOLOT'. We will adhere to this tradition here.

super object type. We say, for instance, that '*man*' '*is_a*' '*person*', that is, each object of type '*man*' is also of type '*person*'.

constraints Structural properties of an object system can be represented using object types, fact types and subtype links, semantic properties will be represented by *constraints*, which are restrictions on relations. We distinguish between *single-relation constraints*, restricting single relations in terms of totality and uniqueness, and *multi-relation constraints*, imposing restrictions on a group of relations.

In BRM, a relationship is divided into two roles representing two ways of looking at the relationship. At the instance level, a relation consists of an ordered pair of objects, one for each role. The object playing the first role is taken from the *domain* of that role, that is, one of the two object types participating in the fact type. The other object is taken from the domain of the second role called the *co-role* of the first role. The two roles in a relationship are each others co-roles. The domain of a role's co-role also is called the *co-domain* of that role. Single-relation constraints impose restrictions on combinations of pairs within a certain relationship. They are represented by special properties of the relationship's roles. A role is said to be *total* if each object from the domain of the role is related to (forms a pair with) some object from the co-domain. A role is said to be *unique* if an object from its domain is related to at most one object in its co-domain. In this case the relation is a function with the unique role as its domain type. If both roles of a relationship are total, the relationship denotes bi-total relations. If one of the roles is total and the other one is unique, the relationship is a *surjection* if both roles are total, an *injection* if both roles are unique, and a *bijection* if it is both a surjection and an injection.

multi-relation constraints We divide multi-relation constraints into *multi-role constraints* and *multi-fact constraints*. Multi-role constraints restrict combinations of objects in different roles, multi-fact constraints restrict combinations of pairs in different relationships. A multi-role constraint is imposed on roles with the same domain, or with domains in the same family,[4] called their *common domain*.[5]

A *totality* constraint states that the objects occurring in the roles together form the set of objects of the common domain of the roles: there is no object in that common domain that occurs in none of the roles.

An *exclusivity* constraint states that objects of the common domain cannot occur in more than one role. In the example schema (see Figure 1), an exclusion constraint is given on roles *wife_of* and *mistress_of*. This means that there is no *woman* (an object of the common domain of *wife_of* and *mistress_of*) occurring in both *wife_of* and *mistress_of*. Note that a combination of totality and exclusivity on a set of roles requires their common domain to be partitioned.

An *equality* constraint states that the set of objects in one of the roles is the same as the set for each role.

[4]We say objects are of the same family if they have a common super object type.

[5]Strictly spoken, their common super object type is the common domain.

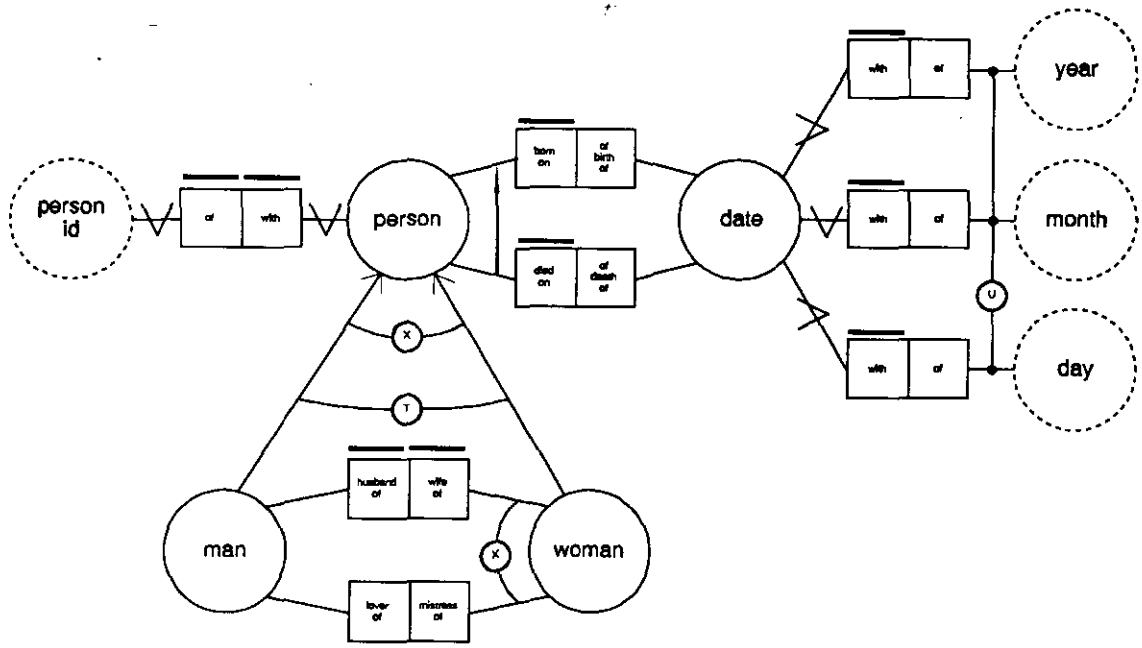

Figure 1: BRM diagram of the running example.

A *subset* constraint from one role to another requires the set of objects occurring in the first role to be a subset of the set of objects occurring in the second role. In the example, a subset constraint is placed on *died_on* and *born_on*, meaning that each *person* occurring in *died_on* should also occur in *born_on*.

A *uniqueness* constraint on a set of roles requires that a tuple of objects, one from each role participating, uniquely identifies an object of their common co-domain. In the example schema, *date* is uniquely identified by a tuple of a *year*, a *month*, and a *day*.

multi-fact constraints There are three multi-fact constraints in BRM: *fact equality*, *fact exclusion* and *fact subset*. Each multi-fact constraint restricts a set of relationships with roles having domains of the same family.

A *fact equality* constraint restricts the set of object pairs occurring in one of the relations to be the same as the set occurring in the other relations. Note that this is more restrictive than two equality constraints on the roles of both facts. While the latter requires only that the domains of the roles should be equal, the former furthermore requires that the pairs formed in the relationship are exactly the same pairs.

A *fact exclusion* constraint requires the set of pairs of one of the relations to be disjoint from the set of pairs of the other relations.

A *fact subset* constraint requires the set of pairs of the first relation to be a subset of the set of pairs of the second relation.

state space A *database state* represents a state of the object system and is specified in BRM by giving the *state* of every object type, LOT or NOLOT—i.e., by giving the representations of the objects of that particular type present in the given state of the object system—and the *state* of every fact type. A database state has to satisfy a number of requirements, which are given in definition A of appendix A. The most important ones are that objects have a valid representation and that the state of a role, as derived from the state of the fact type it belongs to, must be a subset of the state of the corresponding object type. In addition to this, also the graphical constraints must be satisfied. The set of database states satisfying the constraints in definition A is called the *state space*.

Example We will present here as an example conceptual schema the information model of a private investigator. It concerns men and women, who may be married, but also may have extra-marital relations. Note that this schema, the diagram of which is given in Figure 1, can only be a model for a society where married women are faithful to their husbands. It is contrived in order to give a collection of example occurrences of object types, fact types, sub links, and constraints. Thus, the schema is of theoretical value only. The textual specification of this schema can be found in Appendix B.

3 Functional Data Model

The functional data model (FDM) we will use here [ABvH90] is related to a proposal by David Shipman [Shi81]. The basis for Shipman's model was formed by the basic concepts of *objects* or *entities* and *functions* (relations between objects). The essential difference between that model and the functional data model used here is that Shipman's functions are multi-valued in the sense that a function, when applied to an object in its domain, will always yield a *set* of objects. Such a function represents a binary relationship, similar to the fact types in BRM. The functions, as defined in [ABvH90], are always mono-valued, i.e., they yield a single object. The reason for this choice is that, in practice, generally mono-valued functions are encountered. Furthermore, multi-valued functions can always be modeled using mono-valued functions: a mono-valued function when applied inversely also yields a set. A further difference is that Shipman makes no distinction between LOTs and NOLOTs and treats data types, such as string and integer types, needed only for representing names and numbers, as object types. In the functional data model used here, the modeling and representational issues have been separated, just as in the Binary Relationship Model. The FDM *conceptual model* has two components: a structure schema and a representation schema.

structure When constructing an instance of the functional data model for a particular object system, one classifies the objects in the system into *object types*, according to the properties they have. An object type has a label or name and is, mathematically speaking, a set. Properties of objects are given by relating one object to another one, and can be regarded as ordered pairs of objects. The first element of each pair is the object having

the property; the second element is the object giving the details of the property. Because pairs of objects may be related in more than one way, the ordered pairs have to be labeled in order to distinguish the various relations. Properties with the same label are collected into functions, i.e., sets of ordered pairs which are characterized by the fact that the first element of any pair is unique within the set.

Functions are classified into *property types*: sets of functions with the same label. A *structure schema* (see [ABvH90]) specifies which object types are included in the data model and what their properties are, i.e., what kind of functional relationships exist between the various object types. In addition, one can specify a number of graphical constraints, restricting property types to contain only total, injective and/or surjective functions, and object types to be a sub type of another object type, as well as specify key (uniqueness) and mutual exclusion constraints. Other constraints will be specified using the data language [ABvH90]. The choice is relatively arbitrary, representing a trade-off between commonality of the constraints and readability of the diagram. The data language we will use here is a first order language, incorporating the usual mathematical expressions.

representation In the next stage of the modeling process, the representations for the object types are specified in the *representation schema*. The representation of an object type may be structured according to a relational, a network or a hierarchical data model [ABvH90] anticipating the implementation of the database system with a relational, network or hierarchical database management system, respectively. In that case we will have to transform the functional structure schema into the appropriate representation schema. In this paper we have chosen a direct, *functional representation model*, which uses the same basic ingredients for the representations as for the structure schema: sets and functions [ABvH90], which has the virtue of simplicity.

The structure schema is used mostly in the earlier phases of the design process for an information system. The object types occurring in this schema therefore are at the same level as NOLOTs in BRM and entities and relationships in the ER-model [Che76]. The finer details (cf., attributes) are added at a later stage, when the representation schema is constructed by an appropriate transformation of the structure schema and by adding lexical object types and a representation (domain) for every object type. However, one can also choose to include all relevant object types already in the structure schema. In that case only the information concerning representations has to be added to obtain the representation schema. This is the approach followed here.

We will distinguish between *basic* and *derived* representations. In the former case objects are represented directly by associating them with a (possibly complex) value. In the latter case objects are represented by tuples, constructed from the representations of objects from other object types. Every object type may have a basic representation, but only object types for which (total) key constraints have been defined (see [ABvH90]) may have a derived representation.[6] The former type will in general correspond to LOTs,

[6]The total key constraint corresponds to the requirement in BRM that every NOLOT has to be "referenceable".

the latter to NOLOTs in BRM. A typical example of an object type which derives its representation from other object types is an object type modeling a binary [Nij77] or, in general, an n-ary [Che76] relationship between object types. We will see (e.g., in Figure 2) that a fact type in the BRM will be modeled in the FDM by an object type and two property types (one for each role), that connect the object type modeling the fact type to the two object types participating in the fact type. A fact then is identified by specifying two objects, one for each role. On the derivation of the representation of an object type

Figure 2: A binary fact type and its functional equivalent

one has to impose, of course, the restriction that no cycles occur. An object type cannot be represented, directly or indirectly, in terms of itself.

state space A *database state* in the functional model then is specified by giving the *state* of every object type—i.e., by giving the representations of the objects present in the system at that point—and the *state* of their properties. A database state has to satisfy the minimal requirements that objects have a valid representation and that properties in a given database state refer to objects that exist in the same database state. In addition, the graphical constraints, specified in the structure schema, have to be satisfied. The set of database states that satisfy these requirements is called the *state space*. By imposing additional constraints we can restrict the state space to become a *restricted state space*, using a data language.

Example As an example of a structure schema we give the structure diagram (the graphical representation of the structure schema) for our running example (Figure 3).

The notation used is as follows: object types are represented by named boxes; property types by labeled arrows with a dashed shaft and a "feather" at their base.

To express a totality constraint the base half of the arrow representing the property type is drawn with a solid line (Figure 5);

A surjectivity constraint is expressed by making the tip part of the arrow solid (Figure 6).

Injectivity is indicated by omitting the "feather" from the shaft (Figure 7).

Properties satisfying an *is_a* constraint have an *is_a* label in addition to their name (Figure 8). In the diagram properties with the same *is_a* label may be represented by

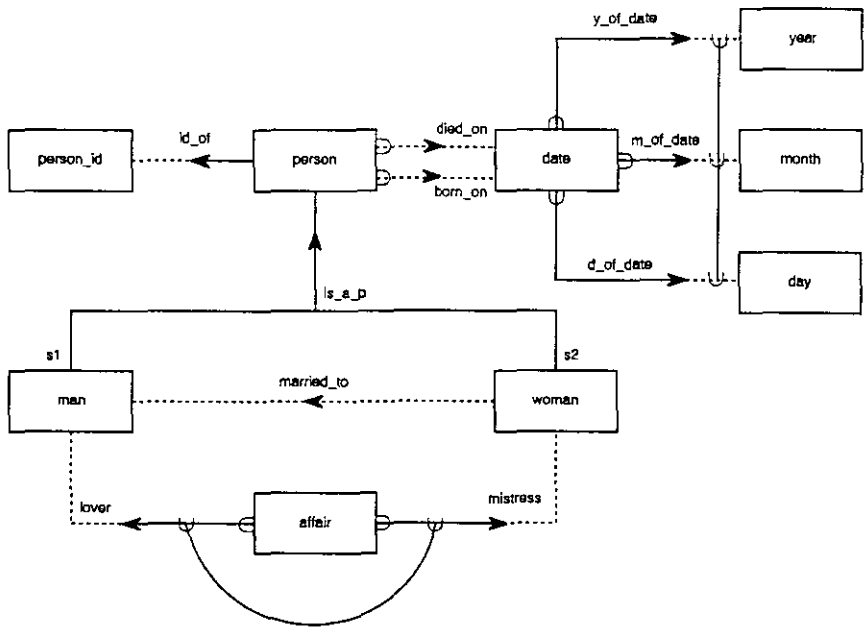

Figure 3: Structure diagram

having the tips of the arrows fuse into one; the corresponding subtypes partition the super type.

In order to be equivalent to the same example represented in the BRM model, the structure diagram has to be supplemented with the following constraints :

- $\mathrm{rng}(s_1) \cup \mathrm{rng}(s_2) = person$
- $\mathrm{rng}(mistress) \cap \mathrm{dom}(married\_to) = \emptyset$
- $\mathrm{dom}(died\_on) \subseteq \mathrm{dom}(born\_on)$

Figure 4: A property type (on the left) and its BRM equivalent

Figure 5: A total property type and its BRM equivalent

Figure 6: A surjective property type and its BRM equivalent

These constraints are expressions in the data language defined in [ABvH90].

In the algorithm, the marriage relation is replaced by an injective property type named *married_to*.

4 The Transformation of BRM to FDM

In this section we will describe the transformation from a BRM-schema to an FDM-schema. We will illustrate the procedure briefly with the running example (see Appendix B).

Figure 7: An injective property type and its BRM equivalent

Figure 8: Subtyping conventions in BRM and FDM

4.1 Approach

We will here give an algorithm to convert each BRM schema into an FDM structure schema and a textual component $L_{C\mathcal{F}}$ representing constraints on the structure schema. We will use a three step approach:

- First, the BRM schema is built up out of the textual specification. In this step, roles are given a unique name.
- Secondly, we will map each concept in the BRM schema onto one or more concepts in an FDM schema in a straightforward way. This is called the mapping phase.
- After that, we will rewrite the FDM schema by deleting redundant properties and object types, and rewriting constraints. This is called the rewriting phase.

In our discussion of the algorithm, we will convert a BRM schema $\mathcal{B}$ to an FDM schema $\mathcal{F}$. The BRM schema $\mathcal{B}$ is a tuple $< L_{\mathcal{B}}, N_{\mathcal{B}}, F_{\mathcal{B}}, R_{\mathcal{B}}, S_{\mathcal{B}}, FT_{\mathcal{B}}, ST_{\mathcal{B}}, C_{\mathcal{B}} >$, with $C_{\mathcal{B}} :=< I_{\mathcal{B}}, A_{\mathcal{B}}, T_{\mathcal{B}}, U_{\mathcal{B}}, E_{\mathcal{B}}, X_{\mathcal{B}}, V_{\mathcal{B}}, CL_{\mathcal{B}} >$.[7] The FDM structure schema $\mathcal{F}$ is a triple $< O_{\mathcal{F}}, P_{\mathcal{F}}, C_{\mathcal{F}} >$ with $P_{\mathcal{F}} :=< F_{\mathcal{F}}, D_{\mathcal{F}}, R_{\mathcal{F}} >$ and $C_{\mathcal{F}} :=< Q_{\mathcal{F}}, U_{\mathcal{F}}, X_{\mathcal{F}} >$. [8]

4.2 The BRM schema.

The input for this step is a well-defined $\mathcal{B}$ schema [NH89]. LOTs are given in $L_{\mathcal{B}}$, NOLOTs in $N_{\mathcal{B}}$. Each fact type is given a unique name if it did not already have one. Each role of a fact type is represented by a name in $R_{\mathcal{B}}$, which is made unique if it was not. The fact type then is represented by its name in $F_{\mathcal{B}}$, and an association with the role names in $FT_{\mathcal{B}}$. Sub links are given unique names: they are represented in $S_{\mathcal{B}}$ and $ST_{\mathcal{B}}$, and for each sub link a pair consisting of a sub type name and super type name is given in $ST_{\mathcal{B}}$. All total role constraints are represented by the names of the constrained roles in $A_{\mathcal{B}}$. The names of the roles that are unique are given in $I_{\mathcal{B}}$. Key constraints are represented in $U_{\mathcal{B}}$ by giving the name of the object type constrained, associated with the names of the roles

[7] See Appendix A.
[8] See e.g. [ABvH90]

identifying the object type. The totality constraints are represented in $T_{\mathcal{B}}$, the exclusivity constraints are given in $X_{\mathcal{B}}$.

In the example schema, only *date* is uniquely identifiable by the *year* of the *date*, its *month* and its *day*. There is only one totality constraint, between sub links. There are two exclusivity constraints: the first one is between the sub links s_1 and s_2, the second one between two roles *mistress_of* and *wife_of*. They are given in $X_{\mathcal{B}}$. The subset constraint between the roles *died_on* and *born_on* is given in $V_{\mathcal{B}}$. Finally, $CL_{\mathcal{B}}$ gives the clustering of sub links s_1 and s_2.

4.3 The Mapping Phase

Definition: We define a mapping function μ in order to map names in BRM to names in FDM:

$\mu(l) :=' L\_' + l.\ l \in L_{\mathcal{B}}$

$\mu(n) :=' N\_' + n,\ n \in N_{\mathcal{B}}$

$\mu(f) :=' F\_' + f,\ f \in F_{\mathcal{B}}$

$\mu(r) :=' R\_' + r,\ r \in R_{\mathcal{B}}$

$\mu(s) :=' S\_' + s,\ s \in S_{\mathcal{B}}$.

Thus, μ is a prefix function. The prefixes enable us to distinguish the collection of object types in $O_{\mathcal{F}}$ in the FDM-structure schema in the rewriting phase. We define μ' as the inverse of μ, that is: μ' 'chops' off the first two characters of names in such a way that $\mu'(\mu(e)) = e$.

Mapping Algorithm: Given this function μ, we will now describe the mapping phase of the transformation algorithm:

- For each LOT type $l \in L_{\mathcal{B}}$, introduce an object type $\mu(l)$ in $O_{\mathcal{F}}$.
- For each NOLOT type $n \in N_{\mathcal{B}}$, introduce an object type $\mu(n)$ in $O_{\mathcal{F}}$.
- For each role $r \in R_{\mathcal{B}}$, introduce a property type $\mu(r)$ in $F_{\mathcal{F}}$.
- For each fact type $f \in F_{\mathcal{B}}$, let $FT_{\mathcal{B}}(f) = ((r_1, o_1), (r_2, o_2))$, where r_1 and r_2 are distinct roles of $R_{\mathcal{B}}$, and o_1 and o_2 are object types. Then:
  - introduce an object type $\mu(f)$ in $O_{\mathcal{F}}$;
  - introduce $(\mu(r_1), \mu(f))$ and $(\mu(r_2), \mu(f))$ in $D_{\mathcal{F}}$;
  - introduce $(\mu(r_1), \mu(o_1))$ and $(\mu(r_2), \mu(o_2))$ in $R_{\mathcal{F}}$;

- $Q_{\mathcal{F}}(\mu(r_1)) := \{< total, \top >, < injective, \bot >, < surjective, \bot >\}$;
- $Q_{\mathcal{F}}(\mu(r_2)) := \{< total, \top >, < injective, \bot >, < surjective, \bot >\}$;
- add $\{< \mu(r_1), \mu(r_2) >\}$ to $U_{\mathcal{F}}(\mu(f))$.

Thus, the fact f, with roles r_1 and r_2 in $R_{\mathcal{B}}$, is mapped to an object type $\mu(f)$ and two total properties $\mu(r_1)$ and $\mu(r_2)$ in $\mathcal{F}$, which together are a key for $\mu(f)$. We say the properties $\mu(r_1)$ and $\mu(r_2)$ 'belong to' the object type $\mu(f)$.

- For each $s \in S_{\mathcal{B}}$, let $ST_{\mathcal{B}}(s) = (sub, super)$:
  - introduce a sub link property $\mu(s)$ in $F_{\mathcal{F}}$;
  - add $(\mu(s), \mu(sub))$ to $D_{\mathcal{F}}$;
  - add $(\mu(s), \mu(super))$ to $R_{\mathcal{F}}$;
  - construct an *is_a* label l; add l to V_{is_a};
  - $Q_{\mathcal{F}}(\mu(s)) := l$.

  Thus, a sub link s in $\mathcal{B}$ is mapped to a property $\mu(s)$ in $\mathcal{F}$, which is labeled as an *is_a* property.

- For $C_{\mathcal{B}}$ do:
  - for each $r \in I_{\mathcal{B}}$: $Q_{\mathcal{F}}(\mu(r)) \cdot injective := \top$;
  - for each $r \in A_{\mathcal{B}}$: $Q_{\mathcal{F}}(\mu(r)) \cdot surjective := \top$;

  Thus, uniqueness constraints on a single role in $\mathcal{B}$ are transferred to injectivity constraints in $\mathcal{F}$; a total role constraint is mapped to surjectivity.
  A totality constraint has to be mapped to a textual constraint, therefore:
  - for each $(o, < r_1, \ldots, r_n >) \in T_{\mathcal{B}}$ add $\mu(o) = \mathrm{rng}(\mu(r_1)) \cup \cdots \cup \mathrm{rng}(\mu(r_n))$ to $L_{C\mathcal{F}}$;

  We will postpone the translation of multi-role uniqueness constraints, because at this stage the property types involved do not have a common domain. This will only be the case after the necessary rewriting of the structure schema resulting from the mapping phase has been done.[9] Therefore, we introduce a temporary set M_U containing translated object types and properties involved in this type of constraint:
  - for each $(o, < r_1, \ldots, r_n >) \in U_{\mathcal{B}}$ add $(\mu(o), < \mu(r_1), \ldots, \mu(r_n) >)$ to M_U;

  For equality, exclusivity, and subset constraints, we have to distinguish between role constraints and fact constraints. Role constraints can easily be translated to range constraints imposed on the properties the roles are translated to. For fact constraints, translation is less easy and an example is given in Figure 9:

[9] Note that for multi-role uniqueness constraints to be meaningful the coroles of the roles involved have to be subject to a uniqueness constraint themselves.

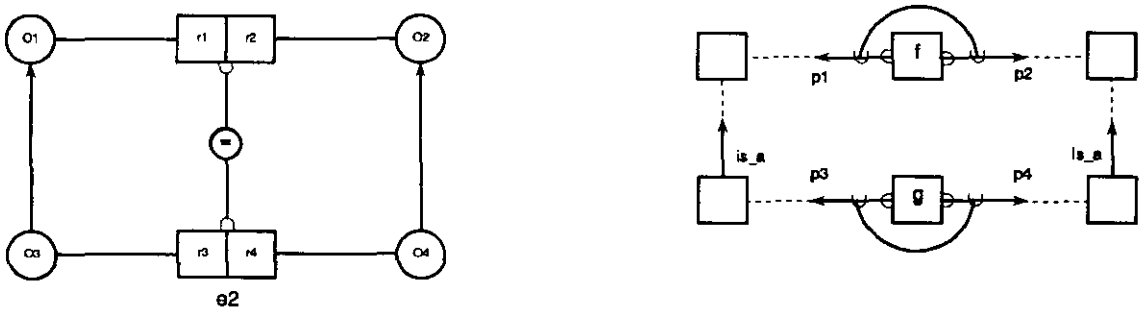

Figure 9: An example of the mapping for an equality constraint on two fact types

- for each $< e_1, e_2 > \in E_{\mathcal{B}}$:
  - if $e_1, e_2 \in R_{\mathcal{B}}$, add $\mathrm{rng}(\mu(e_1)) = \mathrm{rng}(\mu(e_2))$ to $L_{\mathcal{CF}}$;

  In case of a fact equality between e_1 and e_2, we have to express that the set of pairs $(p_1(x), p_2(x))$ of objects representing the state of e_1 in a given database state is equal to the set of pairs $(p_3(y), p_4(y))$ of objects representing the state of e_2 in the same database state (see definition A in appendix A), where $x \in \mu(e_1), y \in \mu(e_2)$, and $p_i = \mu(r_i)$. We define a relation sub between $\mu(e_1)$ and $\mu(e_3)$, and equality in terms of sub:
  - if $e_1, e_2 \in F_{\mathcal{B}}$, let $FT_{\mathcal{B}}(e_1) = \{(r_1, o_1), (r_2, o_2)\}$, $FT_{\mathcal{B}}(e_2) = \{(r_3, o_3), (r_4, o_4)\}$, $f = \mu(e_1), g = \mu(e_2), p_1 = \mu(r_1), p_2 = \mu(r_2), p_3 = \mu(r_3), p_4 = \mu(r_4)$, o_1 and o_3 have a common domain, as well as o_2 and o_4:
    add $\mathrm{sub}(f, g) \wedge \mathrm{sub}(g, f)$ to $L_{\mathcal{CF}}$, where
    $\mathrm{sub}(f, g) := \forall [x : f | \exists [y : g | p_1(x) = p_3(y) \wedge p_2(x) = p_4(y)]]$.
- for each $x \in X_{\mathcal{B}}$:
  In case of exclusivity between roles r_i in $R_{\mathcal{B}}$, an exclusivity constraint is imposed on the ranges of the translated roles, that is, on all $\mu(r_i)$ in $\mathcal{F}$:
  - if x is of the form $< r_1, \ldots, r_n >$, with $r_i \in R_{\mathcal{B}} \cup S_{\mathcal{B}}$:
    - add $(\mu(o), < \mu(r_1), \ldots, \mu(r_n) >)$ to M_X; here o refers to the common domain of r_1 to r_n.
    - add $\bigwedge_{i>j}(\mathrm{rng}(\mu(r_i))) \cap \mathrm{rng}(\mu(r_j)) = \emptyset$, $i, j \in \{1, \ldots, n\}$, to $L_{\mathcal{CF}}$.
  - if x is of the form $< e_1, e_2 >, e_1, e_2 \in F_{\mathcal{B}}$, let p_1 and p_2 be the properties belonging to $f = \mu(e_1)$, and p_3 and p_4 belong to $g = \mu(e_2)$. Add to $L_{\mathcal{CF}}$:
    $\forall [x : \mu(e_1) | \forall y : \mu(e_2)] (p_1(x) = p_3(y)) \rightarrow (p_2(x) \neq p_4(y)) \wedge (p_2(x) = p_4(y)) \rightarrow (p_1(x) \neq p_3(y))]$.
- for each $v \in V_{\mathcal{B}}$:
  - if v is of the form $< p, q >, p, q \in R_{\mathcal{B}}$, add $(\mathrm{rng}(\mu(p)) \subseteq \mathrm{rng}(\mu(q)))$ to $L_{\mathcal{CF}}$.

* if v is of the form $< e_1, e_2 >, e_1, e_2 \in F_{\mathcal{B}}$, add $\mathrm{sub}(\mu(e_1), \mu(e_2))$ to $L_{C\mathcal{F}}$.

A cluster in $\mathcal{B}$ is a set of sub links $s_1, \ldots, s_n$ in $S_{\mathcal{B}}$, associated with the same super type. In an earlier translation, each sub link s_i has already been mapped to a property type $\mu(s_i)$, and an *is_a* label l_i is introduced for it in V_{is_a}. We will associate all those sub links $\mu(s_i)$ with one and the same label, picking the first one of the labels l_i already constructed, and deleting all others.

- for each $(o, < s_1, \ldots, s_n >) \in CL_{\mathcal{B}}$, let l_i range over the associated *is_a* labels such that $Q_{\mathcal{F}}(\mu(s_i)) = l_i$:
  * replace $Q_{\mathcal{F}}(\mu(s_i))$ by l_1;
  * delete $l_i, i \in \{2, \ldots, n\}$ from V_{is_a}.

After the first phase of the algorithm, $O_{\mathcal{F}}$ contains three kinds of object types: each LOT of $\mathcal{B}$ is represented in $O_{\mathcal{F}}$, as well as each NOLOT. Each fact type of $\mathcal{B}$ is also represented by an object type in $O_{\mathcal{F}}$, a 'fact object type'. Roles and sub links in $\mathcal{B}$ are represented by 'role' properties in $\mathcal{F}$ and thus are mapped in $F_{\mathcal{F}}$. The domain of a role property is the fact object type representing the original fact. Its range is the object type in $\mathcal{B}$ associated with the role. The domain of a sub link representation is the representation of the sublink's sub type. Domains and ranges of object types in $\mathcal{F}$ are given in $D_{\mathcal{F}}$ and $R_{\mathcal{F}}$ respectively.

Each fact type of $\mathcal{B}$ introduces a uniqueness constraint in $\mathcal{F}$, imposed on the properties representing the fact's roles. These constraints are represented in $U_{\mathcal{F}}$. $Q_{\mathcal{F}}$ represents totality, injectivity and surjectivity constraints on role properties in $\mathcal{F}$; most of them are derived from total role and unique role constraints of roles in $\mathcal{B}$. Each role property initially is total. M_U represents key constraints of $\mathcal{B}$; in the example case there is only one such constraint. V_{is_a} marks sub links, and $L_{C\mathcal{F}}$ gives the translation of the totality and exclusivity constraints between s_1 and s_2, the exclusivity constraint between *mistress_of* and *wife_of*, and the subset constraint between *died_on* and *born_on*, all in the data language.

4.4 The Rewriting Phase

We now have object types collected in $O_{\mathcal{F}}$, among which are fact object types. Furthermore, we have properties –role properties and sub link properties– collected in $P_{\mathcal{F}}$, with their domains and ranges specified. We will call a role property p *total* iff $Q_{\mathcal{F}}(p) \cdot \mathit{total} = \top$. Analogously, we call p *injective* or *surjective*.

The object types in $\mathcal{F}$ that represent a fact type of the original BRM schema $\mathcal{B}$ are recognizable in $O_{\mathcal{F}}$ because of the labeling function μ. In the rewriting phase, we will try to replace as many of those 'binary object types' as possible by property types according to the correspondence given in Figure 2. This is only possible when one or both of the properties of the 'binary object type' is injective. If none of them is injective, it is not possible to rewrite the part of the schema concerning this object type: the fact object type in $\mathcal{F}$ then represents a pure binary relationship in $\mathcal{B}$. In this phase, we will therefore consider only 'injective' fact types. When we replace a fact object type and its two properties by a

single property, we have to modify all expressions in which any of these appear accordingly. Special care has to be taken when uniqueness or exclusiveness constraints are involved.

We have to perform two steps in the rewriting of a fact object type: first we have to decide which of the two property types will replace the fact object type; next, the constraints in the data language have to be adapted to the new situation. In replacing the fact object type by one of its property types, the constraints of the property type will change accordingly.

- For each fact object type $f \in O_{\mathcal{F}}$, let $FT_{\mathcal{B}}(\mu'(f)) = ((a_i, c_i), (a_j, c_j))$, $p_i = \mu(a_i)$, $p_j = \mu(a_j)$, $r_i = \mu(c_i)$, $r_j = \mu(c_j)$:
  - If exactly one of the property types is injective, say p_i, and $p_i \notin M_U$, we replace f and p_i by an adapted version of p_j:
    - delete f from $O_{\mathcal{F}}$;
    - $D_{\mathcal{F}}(p_j) := R_{\mathcal{F}}(p_i)$;
    - if $p_j \in M_U(r_i)$, add p_j to $U_{\mathcal{F}}(r_i)$ and delete p_j from $M_U(r_i)$;
    - delete $< p_i, f >$ from $D_{\mathcal{F}}$;
    - delete $< p_i, r_i >$ from $R_{\mathcal{F}}$;
    - delete $(f, < p_i, p_j >)$ from $U_{\mathcal{F}}$;
    - $Q_{\mathcal{F}}(p_j) \cdot \mathit{total} := Q_{\mathcal{F}}(p_i) \cdot \mathit{surjective}$;
    - delete $Q_{\mathcal{F}}(p_i)$;
    - replace each occurrence of $\mathrm{rng}(p_i)$ in $L_{C\mathcal{F}}$ by $\mathrm{dom}(p_j)$;
    - delete p_i from $F_{\mathcal{F}}$.
  - If $p_i \in M_U$, then p_j also is *injective*[10] and we proceed as above, switching p_i and p_j.
  - If both p_i and p_j are injective, then, due to the properties of BRM, at most one of r_i, r_j is a LOT type.
    - if one of them is a LOT type, say r_j, we proceed as we did above.
    - if they both are NOLOT types, we have to choose between p_i and p_j:
      - if one of them is surjective, say p_i, we proceed as above. If none or both are surjective, we have to consider multiple role constraints:
      - if one of them is involved in a key constraint $c \in U_{\mathcal{B}}$, say p_j, we proceed as above;
      - otherwise, we consider involvement of $\mu'(p_i)$ and $\mu'(p_j)$ in $T_{\mathcal{B}}$, $E_{\mathcal{B}}$, $X_{\mathcal{B}}$, and $V_{\mathcal{B}}$. If $\mu'(p_i)$ is involved in more constraints than is $\mu'(p_j)$, we proceed as above.
      - In the case we did not come to a decision yet, we proceed as above, making an *ad hoc* decision[11].

[10] It is a property of well-defined BRM schemas that the co-role of a role constrained by uniqueness is unique.

[11] Alternatively, the rewriting system could ask the user's assistance, presenting the alternatives.

- Now that we replaced the fact object type, we have to adapt the data language expressions. For each statement s_i from $L_{C\mathcal{F}} = s_1 \wedge \cdots \wedge s_n$, if s_i is of the form $\mathrm{dom}(p_i) \cap \mathrm{dom}(p_j) = \emptyset$, and provided that $D_{\mathcal{F}}(p_i) = D_{\mathcal{F}}(p_j)$ and p_i and $p_j \in M_X$:
  - add p_i and p_j to $X_{\mathcal{F}}(D_{\mathcal{F}}(p_i))$;
  - delete s_i from $L_{C\mathcal{F}}$.

Finally, all names that obtained a label in the mapping phase are now unlabeled using μ'. Note that all occurrences of names n in the textual constraints of $L_{C\mathcal{F}}$ also have to be replaced by $\mu'(n)$. As a final step property types may be suitably renamed.

In the example schema after the rewriting phase, most of the fact object types have been replaced by properties. The only one remaining is *affair*. Note that at this stage all labels have been removed. For each fact object type removed also a role property has been removed. The domains of the role properties replacing the fact object types have been adapted to the new situation, as well as the constraints in $Q_{\mathcal{F}}$. In $L_{C\mathcal{F}}$, rng(*wife_of*) has been replaced by dom(*husband_of*), because role property *wife_of* and fact object type *matrimony* have been replaced by property *husband_of*.

5 Summary and Future Research

In the previous sections we have seen that it is straightforward to define an instance of a BRM schema in terms of sets of objects and binary relations on sets of objects. We then showed, that it is simple to map a BRM schema to an FDM schema. Because FDM has fewer and simpler graphical constructs than BRM, a number of textual constraints has to be added to the FDM schema in order to preserve equivalence with the BRM schema. In the next step a number of these constraints can be used to simplify the FDM schema, while retaining equivalence.

By mapping a BRM schema to an FDM schema we show that everything which can be modeled using BRM can also be modeled using the functional data model. This then shows the correctness of the interpretation underlying the transformation algorithm. Since FDM has a formal semantics associated with it, the equivalence we established between the models implies a formal semantics for BRM. The semantics for BRM has been worked out in Appendix A. The transformation algorithm has been implemented in Prolog.

Having established a sound basis for interpreting a BRM schema we can now go on and set up a data language. This can be done directly without transforming first to the functional data model. Of course we can benefit from the experience gained by constructing and implementing the functional data language [Hae89]. A data language for BRM would comprise sub languages for defining constraints that cannot be expressed using the graphical conventions (as well as those that can be expressed graphically), for defining queries and updates. The language should not only have the expressive power of, say SQL, but should allow the user also to make use of inheritance of properties, expressed in the model and express recursive queries.

Finally, we note that we have not dealt here with nested or n-ary relations. The inclusion of such modeling constructs in the mapping algorithm is straightforward, since we only have to recognize the fact that a nested relationship is really an object type in its own right and cannot be simplified away in the rewriting phase of the algorithm. The treatment of the constraints becomes a little bit more involved and has also been studied in the context of transforming BRM schemas into relational one [vBtHvdW90]. Alternatively, one can use an algorithm from [NH89] to 'flatten' the relationship first and then apply our algorithm. Similarly, n-ary ($n > 2$) relationships are straightforwardly dealt with: one introduces a 'fact object type' with n instead of two key properties. Such an object type–if strictly n-ary–cannot be simplified in the rewriting phase either. Note that once we have an FDM schema, the transition to a relational schema is well understood (see [ABvH90],[AdBvH92]).

Acknowledgement A concise version of this report has appeared in [AdR91].

References

[ABvH90] A.T.M. Aerts, P.M.E. De Bra, and K.M. van Hee. Combining the Functional and the Relational Model. In A.J. van de Goor, editor, *Proc. of the CSN-90 Conference*, pages 1–16, Utrecht, the Netherlands, 1990.

[AdBvH92] A.T.M. Aerts, P.M.E. de Bra, and K.M. van Hee. Transforming Functional Database Schemes to Relational Representations. In M. Norrie, editor, *Proceedings of the Workshop on Specifications of Database Systems*, Workshops in Computing Science. Springer Verlag, 1992.

[AdR91] A.T.M. Aerts and D. de Reus. Formal Semantics for BRM. In J. van Leeuwen, editor, *Proc. of the CSN-91 Conference*, Utrecht, the Netherlands, 1991.

[AZ90] P. Adriaans and D. Zantinge. *Systeemanalyse.* Academic Service, Schoonhoven, the Netherlands, 1990. (in Dutch).

[Che76] P.P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[dTM86] O. de Troyer and R.A. Meersman. Transforming Conceptual Schema Semantics to Relational Data Applications. In *Information Modelling and Database Management*, . Springer-Verlag, 1986.

[Hab88] H. Habrias. *Le Modèle Relationnel Binaire, Méthode I.A. (NIAM).* Eyrolles, Paris, 1988. (in French).

[Hae89] H.C. Haesen. ELDA Data Manipulatie Taal. In *Computing Science Notes*, number 89/14 in . Eindhoven University of Technology, Eindhoven, the Netherlands, 1989. (in Dutch).

[Mar87] L. Mark. The Binary Relationship Model – 10th Anniversary. Technical Report UMIACS-TR-87-50, CS-TR-1933, University of Maryland, College Park, Maryland 20742, 1987.

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice Hall of Australia Pty Ltd, 1989.

[Nij77] G.M. Nijssen. Current Issues in Conceptual Schema Concepts. In G.M. Nijssen, editor, *Architecture and Models in Database Management Systems*. North Holland Publishing Company, 1977.

[Sen75] M.E. Senko. Information Systems: records, relations, sets, entities, and things. *Information Systems*, 1(1):3–13, 1975.

[Shi81] D.W. Shipman. The Functional Datamodel and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6:140–173, 1981.

[vBtHvdW90] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and Verification of Object-Role Models. Technical Report 90-13, Department of Computing Science, Nijmegen University, Nijmegen, the Netherlands, November 1990.

[vHSV89] K.M. van Hee, L.J. Somers, and M. Voorhoeve. Executable specifications for distributed information systems. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*, , pages 139–155. Elseviers Science Publishing B.V., North-Holland, 1989.

[Win87] J.J.V.R. Wintraecken. *Informatie-analyse volgens NIAM, in theorie en praktijk*. Academic Service, Schoonhoven, the Netherlands, 1987. (in Dutch).

A Definitions for BRM

Definition 1. *Binary Relationship Schema*
A binary relationship schema [dTM86] is a tuple $< L, N, F, R, S, FT, ST, C >$ with:

- L a finite set of lexical object type names.
- N a finite set of non-lexical object type names. We will use $O = L \cup N$ to denote the set of all object type names.

F a finite set of fact type names.

R a finite set of role names. Every role $r \in R$ has a *co-role* associated with it : $\bar{r} \in R$ and $\bar{r} \neq r$. It holds that $\{ \bar{r} \mid r \in R\} = R$

S a finite set of sub link names.

L, N, F, R, and S are mutually disjoint.

FT a finite set of facts. For each $f \in F$, $FT(f) \in R \rightarrow N \cup L$, and $|FT(f)| = 2$. We have $\bigcup_{f \in F} \mathrm{dom}(\mathrm{FT}(f)) = R$ and $\forall f, f' \in F \ : \ f \neq f' \rightarrow \mathrm{dom}(\mathrm{FT}(f)) \cap \mathrm{dom}(\mathrm{FT}(f') = \emptyset$.

ST a finite set of sub links. For each $s \in S$, $ST(s) \in N \times N$.

C a finite set of constraints given by a tuple $< I, A, T, U, E, X, V, CL >$ with:

- I a finite set of role names $r_1, \ldots, r_n$, defining the set of identifying roles.
- A a finite set of role names $r_1, \ldots, r_n$, defining the set of total roles.
- T a finite set of totality constraints. Each totality constraint is denoted as $(o, < r_1, \ldots, r_n >)$, $n \geq 2$, $o \in N$, $r_i \in R$ and $\exists f \in F$ such that $(o, r_i) \in FT(f)$ for $i \in [1..n]$.
- U a finite set of uniqueness constraints analogous to T.
- E a finite set of equality constraints. Each equality constraint is either denoted as $< r_1, r_2 >, r_1, r_2 \in R$, or as $< f_1, f_2 >, f_1, f_2 \in F$. In the first case the roles refer to the same object type, in the second case the fact types refer to the same pair of object types.
- X a finite set of exclusion constraints. Each exclusion constraint is either denoted as $< r_1, \ldots, r_n >$, $n \geq 2$, where $r_i \in R \cup S$, or as $< f_1, f_2 >$, $f_1, f_2 \in F$. Again the corresponding object types have to match.
- V a finite set of subset constraints. Each subset constraint is either denoted as a pair $< r_1, r_2 >, r_1, r_2 \in R$, or as a pair $< f_1, f_2 >, f_1, f_2 \in F$ and the object types involved have to match.
- CL a finite set of clusters. Each cluster is of the form $(o, < s_1, \ldots, s_n >)$, where $o \in N, s_i \in S$ and $\forall i \in [1..n]$: $(sub_i, o) \in ST(s_i)$.

Definition 2. *Domain Function*
A *domain function* for a binary relationship schema $< L, N, F, R, S, FT, ST, C >$ is a function DOM with the following properties:

- $\mathrm{dom}(\mathrm{DOM}) = N \cup L \cup F$
- for $n \in N$: $\mathrm{DOM}(n)$ is a set of *objects*, called the *domain* of n

- for $l \in L$: DOM(l) is a set of *values*, called the *domain* of l

- for $f \in F$: for $(f, \{(r_1, o_1), (r_2, o_2)\}) \in FT$ we have
  $\text{DOM}(f) = \prod(\{(r_1, \text{DOM}(o_1)), (r_2, \text{DOM}(o_2))\})$.

  DOM(f) is a set of two-tuples; for $t \in \text{DOM}(f)$ we have $t(r_i) \in \text{DOM}(o_i)$ for $i \in \{1,2\}$.

- for $s \in S$: for $(sub, super) = \text{ST}(s)$ we have $sub, super \in N$ and $\text{DOM}(sub) = \text{DOM}(super)$.

Definition 3. *Conceptual Model*
A *conceptual model* is a pair consisting of a binary relationship schema
$< L, N, F, R, S, FT, ST, C >$ and a corresponding domain function DOM.

Definition 4. *Database State*
Given a conceptual model $\ll L, N, F, R, S, FT, ST, C >$, DOM> a *database state* is a function b such that:

1. $\text{dom}(b) = O \cup F \cup R$
2. for $o \in O$: $b(o) \subseteq \text{DOM}(o)$ and $b(o)$ is finite.
3. for $f \in F$ and $(f, \{(r_1, o_1), (r_2, o_2)\}) \in \text{FT}$ we have
   - $b(f) \subseteq \prod(\{(r_1, b(o_1)), (r_2, b(o_2))\})$
   - $b(r_i) = \{t(r_i) \mid t \in b(f)\}$, so $b(r_i) \subseteq b(o_i)$ for $i \in \{1,2\}$

   $b(f)$ is a set of two-tuples; for $t \in b(f)$ we have $t(r_i) \in b(o_i)$ for $i \in \{1,2\}$
4. for $s \in S$ and $(sub, super) = \text{ST}(s)$: $b(sub) \subseteq b(super)$
5. $\forall r \in I$ and $f \in F$ such that $r \in \text{dom}(\text{FT}(f))$ we have
   $\forall t, t' \in b(f)$: $t(r) = t'(r) \rightarrow t = t'$
6. $\forall r \in A$: for $f \in F$ such that $(r, o) \in \text{FT}(f)$: $b(r) = b(o)$.
7. $\forall (o, < r_1, \ldots, r_n >) \in T$: $\bigcup_{i \in [1..n]} b(r_i) = b(o)$
8. $\forall (ot, < r_1, \ldots, r_n >) \in U$: $\forall o, o' \in b(ot)$:
   $(\forall i \in [1..n] \exists t_i, t_i' \in b(f_i)$: $(t_i(\bar{r}_i) = o \wedge t_i'(\bar{r}_i) = o') \wedge (t_i(r_i) = t_i'(r_i))) \rightarrow o = o'$,
   where $f_i \in F$, such that $(r_i, ot) \in \text{FT}(f_i)$ for $i \in [1..n]$
9. $\forall < r_1, r_2 > \in E, r_1, r_2 \in R$ it holds that $b(r_1) = b(r_2)$, and $\forall < f_1, f_2 > \in E, f_1, f_2 \in F$ it holds that $b(f_1) = \grave{b}(f_2)$, where $\grave{b}(f_2)$ is obtained from $b(f_2)$ by replacing in every tuple of $b(f_2)$ the role-names with the corresponding role-names from f_1.

10. $\forall r_1, \ldots, r_n \in X, r_i \in R \cup S$ we have $\forall_{i,j \in [1..n], i \neq j} b(r_i) \cap b(r_j) = \emptyset$.
    $\forall < f_1, f_2 > \in X, f_1, f_2 \in F$ we require that $b(f_1) \cap \tilde{b}(f_2) = \emptyset$, where $\tilde{b}(f_2)$ as above.

11. $\forall < r_1, r_2 > \in V, r_1, r_2 \in R$ it holds that $b(r_1) \subseteq b(r_2)$, and $\forall < f_1, f_2 > \in V, f_1, f_2 \in F$ it holds that $b(f_1) \subseteq \tilde{b}(f_2)$, where $\tilde{b}(f_2)$ as above.

12. $\forall (o, sub_1, \ldots, sub_n) \in CL, o \in N$ and $s_i \in S$ such that for $i \in [1..n]$: $(sub_i, o) = \mathrm{ST}(s_i)$, we have $\forall_{i,j \in [1..n], i \neq j} b(sub_i) \cap b(sub_j) = \emptyset$.

B The running example.

We here give the textual BRM specification of the running example, as well as the translation and rewriting of it into an FDM schema.

B.1 Textual Specification

```
"Information concerning dates:"
LOT year, month, day.
NOLOT date.
FACT WITH ROLE with ON NOLOT date AND ROLE of ON LOT year.
EACH date with year.                    # totality constraint on with
date with AT MOST ONE year.             # uniqueness constraint on with
FACT WITH ROLE with ON NOLOT date AND ROLE of ON LOT month.
EACH date with month.
date with AT MOST ONE month.
FACT WITH ROLE with ON NOLOT date AND ROLE of ON LOT day.
EACH date with day.
date with AT MOST ONE day.
date IDENTIFIED BY year of date, month, day.
                                        # key constraint on date

"Information concerning persons:"
LOT person_id.
NOLOT person.
FACT WITH ROLE belongs_to ON NOLOT person_id
    AND ROLE has ON NOLOT person.
person_id belongs_to AT MOST ONE person.
EACH person has person_id.
person has AT MOST ONE person_id.
NOLOT man.
SUBLINK s1 FROM man TO person.
NOLOT woman.
SUBLINK s2 FROM woman TO person.
```

```
EACH person IS A man OR IS A woman.
man, woman ARE MUTUAL EXCLUSIVE.
FACT matrimony WITH ROLE husband_of ON NOLOT man
    AND ROLE wife_of ON NOLOT woman.
man husband_of AT MOST ONE woman.
woman wife_of AT MOST ONE man.
CLUSTER c1 OF s1, s2.

"dangerous (homogeneous) liaisons:"
FACT affair WITH ROLE lover_of ON NOLOT man
    AND ROLE mistress_of ON NOLOT woman.

"women are faithful:"
woman mistress_of man, woman wife_of man ARE MUTUAL EXCLUSIVE.

"Interesting relations and some constraints:"
FACT WITH ROLE died_on ON NOLOT person
    AND ROLE of_death_of ON NOLOT date.
person died_on AT MOST ONE date.
FACT WITH ROLE born_on ON NOLOT person
    AND ROLE of_birth_of ON NOLOT date.
person born_on AT MOST ONE date.
person died_on date SUBSET OF person born_on date.
```

B.2 The Translation

We will here give the transformation of the example schema.

B.2.1 The BRM schema.

The example schema is given in the BRM schema $\mathcal{B} =< L_{\mathcal{B}}, N_{\mathcal{B}}, F_{\mathcal{B}}, R_{\mathcal{B}}, S_{\mathcal{B}}, FT_{\mathcal{B}}, ST_{\mathcal{B}}, C_{\mathcal{B}} >$, with $C_{\mathcal{B}} =< I_{\mathcal{B}}, A_{\mathcal{B}}, U_{\mathcal{B}}, T_{\mathcal{B}}, E_{\mathcal{B}}, X_{\mathcal{B}}, V_{\mathcal{B}}, CL_{\mathcal{B}} >$, and

$L_{\mathcal{B}}$: $\{year, month, day, person\_id\}$;

$N_{\mathcal{B}}$: $\{date, person, man, woman\}$;

$F_{\mathcal{B}}$: $\{f_1, f_2, f_3, f_4, matrimony, affair, f_5, f_6\}$

$R_{\mathcal{B}}$: $\{with_1, of_1, with_2, of_2, with_3, of_3, belongs\_to, has, husband\_of, wife\_of, lover\_of, mistress\_of, died\_on, of\_death\_of, born\_on, of\_birth\_of\}$

$S_{\mathcal{B}}$: $\{s_1, s_2\}$

$FT_{\mathcal{B}}$: $\{(f_1,((with_1,date),(of_1,year))),(f_2,((with_2,date),(of_2,month))),$
$(f_3,((with_3,date),(of_3,day))),(f_4,((belongs\_to,person\_id),(has,person))),$
$(matrimony,((husband\_of,man),(wife\_of,woman))),$
$(affair,((lover\_of,man),(mistress\_of,woman))),$
$(f_5,((died\_on,person),(of\_death\_of,date))),(f_6,((born\_on,person),(of\_birth\_of,date)))\}$

$ST_{\mathcal{B}}$: $\{(s_1,(man,person)),(s_2,(woman,person))\}$

$I_{\mathcal{B}}$: $\{with_1,with_2,with_3,belongs\_to,has,husband\_of,wife\_of,born\_on,died\_on\}$

$A_{\mathcal{B}}$: $\{with_1,with_2,with_3,has\}$

$U_{\mathcal{B}}$: $\{(date,<of_1,of_2,of_3>)\}$

$T_{\mathcal{B}}$: $\{(person,<s_1,s_2>)\}$

$E_{\mathcal{B}}$: $\{\}$

$X_{\mathcal{B}}$: $\{<s_1,s_2>,<mistress\_of,wife\_of>\}$

$V_{\mathcal{B}}$: $\{<died\_on,born\_on>\}$

$CL_{\mathcal{B}}$: $\{(c_1,<s_1,s_2>)\}$

B.2.2 The mapped schema.

$O_{\mathcal{F}}$: $\{L\_year,L\_month,L\_day,L\_person\_id,N\_date,N\_person,N\_man,N\_woman,$
$F\_f_1,F\_f_2,F\_f_3,F\_f_4,F\_matrimony,F\_affair,F\_f_5,F\_f_6\};$

$F_{\mathcal{F}}$: $\{R\_with_1,R\_with_2,R\_with_3.R\_of_1,R\_of_2,R\_of_3,R\_belongs\_to,R\_has,$
$R\_husband\_of,R\_wife\_of,R\_lover\_of,R\_mistress\_of,$
$R\_died\_on,R\_of\_death\_of,R\_born\_on,R\_of\_birth\_of,$
$S\_s_1,S\_s_2\}$

$D_{\mathcal{F}}$: $\{<R\_with_1,F\_f_1>,<R\_with_2,F\_f_2>,<R\_with_3,F\_f_3>,$
$<R\_of_1,F\_f_1>,<R\_of_2,F\_f_2>,<R\_of_3,F\_f_3>,$
$<R\_belongs\_to,F\_f_4>,<R\_has,F\_f_4>,$
$<R\_husband\_of,F\_matrimony>,<R\_wife\_of,F\_matrimony>,$
$<R\_lover\_of,F\_affair>,<R\_mistress\_of,F\_affair>,$
$<R\_died\_on,F\_f_5>,<R\_of\_death\_of,F\_f_5>,$
$<R\_born\_on,F\_f_6>,<R\_of\_birth\_of,F\_f_6>,$
$<S\_s_1,N\_man>,<S\_s_2,N\_woman>\}$

$R_{\mathcal{F}}$: $\{<R\_with_1,N\_date>,<R\_with_2,N\_date>,<R\_with_3,N\_date>,$
$<R\_of_1,L\_year>,<R\_of_2,L\_month>,<R\_of_3,L\_day>,$
$<R\_belongs\_to,L\_person\_id>,<R\_has,N\_person>,$
$<R\_husband\_of,N\_man>,<R\_wife\_of,N\_woman>,$

$< R\_lover\_of, N\_man >, < R\_mistress\_of, N\_woman >,$
$< R\_died\_on, N\_person >, < R\_of\_death\_of, N\_date >,$
$< R\_born\_on, N\_person >, < R\_of\_birth\_of, N\_date >,$
$< S\_s_1, N\_person >, < S\_s_2, N\_person >\}$

V_{is_a}: $\{l_1\}$

$Q_{\mathcal{F}}$: $\{< R\_with_1, (\top, \top, \top) >, < R\_with_2, (\top, \top, \top) >, < R\_with_3, (\top, \top, \top) >,$
$< R\_of_1, (\top, \bot, \bot) >, < R\_of_2, (\top, \bot, \bot) >, < R\_of_3, (\top, \bot, \bot) >,$
$< R\_belongs\_to, (\top, \top, \bot) >, < R\_has, (\top, \top, \top) >,$
$< R\_husband\_of, (\top, \top, \bot) >, < R\_wife\_of, (\top, \top, \bot) >,$
$< R\_lover\_of, (\top, \bot, \bot) >, < R\_mistress\_of, (\top, \bot, \bot) >,$
$< R\_died\_on, (\top, \top, \bot) >, < R\_of\_death\_of, (\top, \bot, \bot) >,$
$< R\_born\_on, (\top, \top, \bot) >, < R\_of\_birth\_of, (\top, \bot, \bot) >,$
$< S\_s_1, l_1 >, < S\_s_2, l_1 >\}$

$U_{\mathcal{F}}$: $\{(F\_f_1, < R\_with_1, R\_of_1 >), (F\_f_2, < R\_with_2, R\_of_2 >),$
$(F\_f_3, < R\_with_3, R\_of_3 >), (F\_f_4, < R\_belongs\_to, R\_has >),$
$(F\_matrimony, < R\_husband\_of, R\_wife\_of >),$
$(F\_affair, < R\_lover\_of, R\_mistress\_of >),$
$(F\_f_5, < R\_died\_on, R\_of\_death\_of >), (F\_f_6, < R\_born\_on, R\_of\_birth\_of >)\}$

M_U: $\{(N\_date, < R\_of_1, R\_of_2, R\_of_3 >)\}$

$L_{C\mathcal{F}}$: $N\_person = \mathrm{rng}(S\_s_1) \cup \mathrm{rng}(S\_s_2) \ \wedge$
$\mathrm{rng}(S\_s_1) \cap \mathrm{rng}(S\_s_2) = \emptyset \ \wedge$
$\mathrm{rng}(R\_mistress\_of) \cap \mathrm{rng}(R\_wife\_of) = \emptyset \ \wedge$
$\mathrm{rng}(R\_died\_on) \subseteq \mathrm{rng}(R\_born\_on)$

B.2.3 The rewritten schema.

The resulting schema is $\mathcal{F} =< O_{\mathcal{F}}, P_{\mathcal{F}}, C_{\mathcal{F}} >$, where $P_{\mathcal{F}} =< F_{\mathcal{F}}, D_{\mathcal{F}}, R_{\mathcal{F}} >$ and $C_{\mathcal{F}} =< Q_{\mathcal{F}}, U_{\mathcal{F}}, X_{\mathcal{F}} >$, and $L_{C\mathcal{F}}$ are given by:

$O_{\mathcal{F}}$: $\{year, month, day, person\_id, date, person, man, woman, affair, \};$

$F_{\mathcal{F}}$: $\{of_1, of_2, of_3, belongs\_to, husband\_of, lover\_of, mistress\_of,$
$of\_death\_of, of\_birth\_of, s_1, s_2\}$

$D_{\mathcal{F}}$: $\{< of_1, date >, < of_2, date >, < of_3, date >, < belongs\_to, person >,$
$< husband\_of, woman >, < lover\_of, affair >, < mistress\_of, affair >,$
$< of\_death\_of, person >, < of\_birth\_of, person >,$
$< s_1, man >, < s_2, woman >\}$

$R_{\mathcal{F}}$: $\{< of_1, year >, < of_2, month >, < of_3, day >,$
$< belongs\_to, person\_id >, < husband\_of, man >,$
$< lover\_of, man >, < mistress\_of, woman >,$
$< of\_death\_of, date >, < of\_birth\_of, date >,$
$< s_1, person >, < s_2, person >\}$

V_{is_a}: $\{l_1\}$

$Q_{\mathcal{F}}$: $\{< of_1, (\top, \bot, \bot) >, < of_2, (\top, \bot, \bot) >, < of_3, (\top, \bot, \bot) >,$
$< belongs\_to, (\top, \top, \bot) >, < husband\_of, (\bot, \top, \bot) >,$
$< lover\_of, (\top, \bot, \bot) >, < mistress\_of, (\top, \bot, \bot) >,$
$< of\_death\_of, (\bot, \bot, \bot) >, < of\_birth\_of, (\top, \bot, \bot) >, < s_1, l_1 >, < s_2, l_1 >\}$

$U_{\mathcal{F}}$: $\{(affair, < lover\_of, mistress\_of >), (date, < of_1, of_2, of_3 >)\}$

$X_{\mathcal{F}}$: $\{\}$

$L_{C\mathcal{F}}$: $\{person = \mathrm{rng}(s_1) \cup \mathrm{rng}(s_2),$
$\mathrm{rng}(s_1) \cap \mathrm{rng}(s_2) = \emptyset,$
$\mathrm{rng}(mistress\_of) \cap \mathrm{dom}(husband\_of) = \emptyset,$
$\mathrm{dom}(of\_death\_of) \subseteq \mathrm{dom}(of\_birth\_of)\}$

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a parallel program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.