Semantics and Verification of Object-Role Models

P. van Bommel
A.H.M. ter Hofstede *
Th.P. van der Weide †

April 29, 1993

Published as: P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and Verification of Object-Role Models. *Information Systems*, 16(5):471–495, October 1991.

Abstract

In this paper we formalise data models that are based on the concept of predicator, the combination of an object type and a role. A very simple model, the Predicator Model, is introduced in a rigid formal way.

We introduce the concept of population as an instantiation of an information structure. A primitive manipulation language is defined in the style of Relational Algebra. Well-known types of constraints are defined in terms of the algebra introduced, as restrictions on populations. They are given more expressive power than is usually the case.

Constraints are of central importance for identification purposes. Weak identification ensures identifiability of objects within a specific population, while structural identification ensures identifiability of objects within every population.

Different levels of constraint inconsistency are defined and it is shown that the verification of two important levels is NP-complete.

1 Introduction

Nowadays many techniques exist for the modelling of information structures. Some often used techniques have an underlying object-role model. Examples are: NIAM ([15], [30], [25]), the ER Approach ([3], [21]), and functional data models (e.g. FDM [6]). The information structure forms the kernel of the so-called conceptual model, and is considered an important milestone during the process of information analysis. Currently, workbenches are built that support the information analyst in the construction of the conceptual model. The next step consists of the (automated) transformation of the model into an efficient implementation.

In this paper we define a platform, called the Predicator Model, for object-role models. The Predicator Model can be used to specify information structures. Relational operators are introduced that can be used to express the semantics of (complex) constraints. Various types of constraints are usually expressed graphically. We focus on these constraints, that allow us to prove some important properties, such as identifiability.

With respect to the graphical representation of information structures, we make use of the drawing conventions of NIAM. This technique does not restrict itself to binary relationships, but allows for relationships of arbitrary degree. Another important feature is that relationships can

*This work has been partially supported by SERC project SOCRATES. Software Engineering Research Centre (SERC), P.O. Box 424, 3500 AK Utrecht, The Netherlands, E-mail: hofstede@serc.nl

†This work has been partially supported by ESPRIT project APPED (2499). Dept. of Information Systems, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, E-mail: tvdw@cs.kun.nl

be objectified and thus be treated as objects. There is no distinction made between entities and attributes (as in the ER Approach), only between so-called lexical and non-lexical objects. Finally, in NIAM many types of constraints can be expressed graphically.

Up to now there does not exist a formal specification of object-role data models. Some attempts have been made (e.g. [5], [19], [22]), however they are not complete. This lack of formality results in several problems. Firstly, ambiguity may arise. Different analysts may have different ideas about the meaning of a certain model, thus increasing the possibility of an erroneous design. Secondly, verification of models is hardly possible. Current CASE-tools that support object-role data models, such as RIDL* ([23]), SDW ([16]), ADDS ([20]), ProNIAM ([4]), can only detect some syntactical flaws. They are not capable of detecting all semantical inconsistencies in a model specified by the information analyst. Thirdly, properties of object-role data modelling techniques or of the models expressed in such a technique, can not be proven. Comparison with other techniques or data models is impossible on a formal basis. For a more detailed discussion on these issues, see [11].

The organisation of this paper is as follows. First we introduce the Predicator Model. The Predicator Model is used to describe an information structure. The meaning of an information structure is described by the concept of instantiation (or population). We make a distinction between the structure and its instantiations: instantiations are *not* part of the information structure. Next we introduce, in the style of relational algebra, a primitive language to reason about the Predicator Model. Relational expressions describe relations that can be derived from a given information structure. As a consequence, a relational expression describes the type of the derived relation, and gives a rule how its instantiation (population) can be derived from a given population of the information structure. An advantage of relational algebra is that it is sufficiently elementary to allow the description of more sophisticated languages (for example RIDL and SQL). Relational expressions (derived fact types) form the basis for describing the semantics of static constraints. We introduce a powerful set of constraint types. The so-called total role constraint and the uniqueness constraint turn out to be rather complex, when applied sophisticatedly (for example uniqueness constraint in combination with objectification). We introduce the Uniquest Algorithm as the semantics for uniqueness constraints. In the next section we focus on identification of object occurrences within populations. A minimal requirement is weak identification. On the schema level however, we have structural identification, which is powerful enough to guarantee weak identification for any population. After this we consider properties of schemas, mainly in the context of checking schema consistency. Simple properties are easily checked, and will detect a major part of usual errors made by (unexperienced) information system analysts. Complexity of checking schema properties is addressed in the last section. We show that the relevant properties are NP-complete problems.

2 The Predicator Model

2.1 Introduction

In this section we will introduce the notion of *relation type* as is usual in the world of databases, where it is also denoted as *fact type*. The term relation suggests a connection with the corresponding mathematical notion. In classic approaches, this leads to the definition of a relation as a subset of a Cartesian product. This approach is known as the *tuple oriented approach*. A disadvantage of this approach is that algebraic operators lack useful properties as commutativity and associativity. The modern approach is to use the mapping mechanism to describe relations (the *mapping oriented approach*). See also [13].

In ER and NIAM a relation type is considered to be an association between object types. The graphical representation of a relation type, drawn in the NIAM style, is shown in figure 1. The corresponding ER diagram is shown in figure 2. We prefer the NIAM style for the rest of this paper, as this makes the concept of role more explicit. A role denotes a particular function that is played by an object type in a relation type.

In figure 1 we see that the roles r_1 and r_2 are ordered. The mapping oriented approach

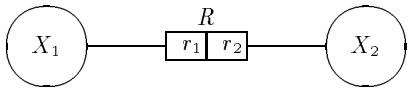


Figure 1: A NIAM relation type

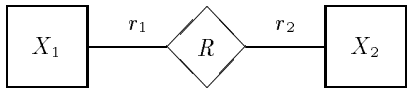


Figure 2: The corresponding ER diagram

corresponds to disconnecting the relation R, yielding the situation of figure 3. Note that R is represented in this figure according to the same format as X_1 and X_2. In this figure, the basic building element is the connection between an object type and a role, the so-called *predicator* (this term was first introduced in [7]). In figure 3, p_1 is the predicator connecting X_1 to r_1, and p_2 the predicator that connects X_2 to r_2. Note that the concept of predicator appears itself in a NIAM schema (as an unnamed drawing object), while in an ER schema it cannot be visualised.

We go a step further, and consider a relation type as a set of predicators. A direct consequence is that we consider a relation type as an association between predicators, rather than between object types.

We will not take the full consequence of this when drawing information structures. We will prefer the style of figure 1 or 3 rather than the style of figure 4 or figure 5. In figure 4 object types are drawn in a nested fashion. The functional drawing style from figure 5 abstracts from the internal structure of relation type R. This internal structure is captured by the direction of the arrows that represent the predicators. This style will be very useful to describe complex operations on a schema.

2.2 The information structure

The discussion in the previous section leads to the following definition: an *information structure* is a structure consisting of the following basic components:

1. A finite set $\mathcal{P}$ of *predicators*.
2. A set $\mathcal{O}$ of *object types*.
3. A partition $\mathcal{F}$ of the set $\mathcal{P}$. The elements of $\mathcal{F}$ are called *fact types*.

Object types are classified as follows. First we have: $\mathcal{F} \subseteq \mathcal{O}$. The object types in $\mathcal{F}$ are the *composed object types*, the object types in $\mathcal{O} \setminus \mathcal{F}$ are the *atomic object types* ($\mathcal{A}$). There are two different sorts of atomic object types: entity types ($\mathcal{E}$) and label types ($\mathcal{L}$). The difference is that labels can, in contrast with entities, be represented (reproduced) on a communication medium. Depending on the medium, we distinguish text, graphics, sound and video. The term multimedia is used as a collective noun.

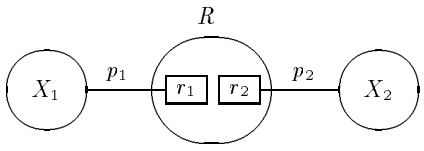


Figure 3: Disconnecting the roles

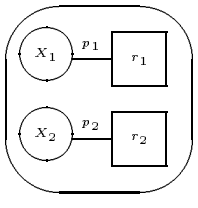


Figure 4: A relation type as a set of predicators

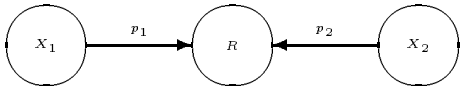


Figure 5: The functional drawing style

It is important to note that instances (occurrences) of object types are *not* within the information structure. Instantiations (populations) will be introduced in a later section.

2.3 Subtyping

The concept of subtype is defined as a partial order Sub on atomic object types (Sub $\subseteq \mathcal{E}\times\mathcal{E}\cup\mathcal{L}\times\mathcal{L}$), with the convention that a Sub b is interpreted as: a is a subtype of b. This partial order should have the property that with each element of $\mathcal{A}$ a (unique) top element can be associated. This pater familias is found by the function $\sqcap : \mathcal{A} \rightarrow \mathcal{A}$ (which is similar to the top operator from lattice theory). Each subtype has a subtype defining rule. This will be introduced in section 4.7.

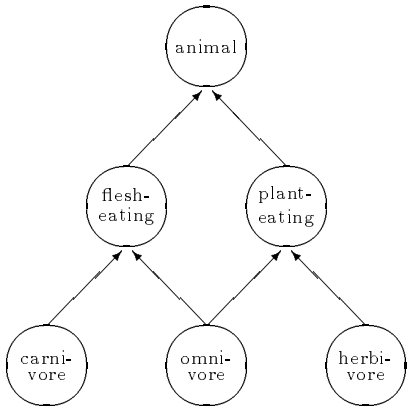


Figure 6: Example of a subtype hierarchy

Example 2.1 *In figure 6 we have the following subtype hierarchy:*

flesh-eating animal Sub *animal*

plant-eating animal Sub *animal*

carnivore Sub *flesh-eating animal*

omnivore Sub *flesh-eating animal*

omnivore Sub *plant-eating animal*

herbivore Sub *plant-eating animal*

Each subtype relation is represented as an arrow in figure 6. As a consequence, the pater familias of object type carnivore *is* animal, *or:* $\sqcap$(carnivore) = animal.

Remark 1: Subtype requirements at the fact level are expressed by subset constraints, that are introduced in a later section.

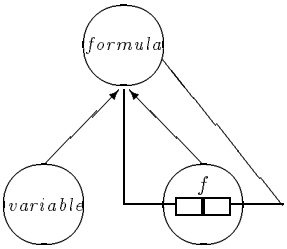


Figure 7: Example of disjoint union

Remark 2: In this approach, subtyping is a mechanism for refining atomic object types. It does not provide such a thing as *disjoint union*. In programming languages this concept is used as a construction mechanism, to describe data types such as *formula*. A *formula* may be either a single variable, or constructed by some function (say f) from simpler formulas. This is displayed in figure 7. The usefulness of the disjoint union in information structures has been discussed in [28].

2.4 Basics

The base of a predicator is the object part of that predicator. The associated object type is found by the following elementary operator:

$$\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$$

We call predicators p and q *attached* to each other ($p \sim q$), when $\sqcap(\mathsf{Base}(p)) = \sqcap(\mathsf{Base}(q))$. The fact type that corresponds with a predicator is obtained by the operator:

$$\mathsf{Fact} : \mathcal{P} \rightarrow \mathcal{F}$$

which is defined by: $\mathsf{Fact}(p) = f \Leftrightarrow p \in f$.

Usually (NIAM, ER), relation types are restricted to *fact types* and *reference types*. A reference type (or attribute type) is a binary relation between an entity type and a label type, i.e. a set of predicators $\{p, q\}$, where $\mathsf{Base}(p) \notin \mathcal{L}$ and $\mathsf{Base}(q) \in \mathcal{L}$. A fact type is a relation type between entity types only, i.e. a set of predicators f, such that: $\forall_{p \in f}\,[\mathsf{Base}(p) \notin \mathcal{L}]$. In this paper, we will use the term *fact type* as a generic term for relation types in information structures.

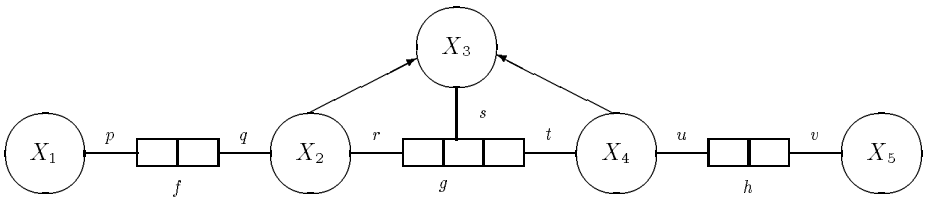


Figure 8: Example information structure

Example 2.2

In figure 8 we see an example of an information structure. We have:

$$\begin{aligned} \mathcal{P} &= \{p, q, r, s, t, u, v\} \\ \mathcal{O} &= \{X_1, X_2, X_3, X_4, X_5, f, g, h\} \\ \mathcal{F} &= \{f, g, h\} \end{aligned}$$

where $f = \{p, q\}$, $g = \{r, s, t\}$ and $h = \{u, v\}$. With respect to the predicators we have:

$$\begin{array}{lcllcl} \mathsf{Base}(p) & = & X_1 & \mathsf{Fact}(p) & = & f \\ \mathsf{Base}(q) & = & X_2 & \mathsf{Fact}(q) & = & f \\ \mathsf{Base}(r) & = & X_2 & \mathsf{Fact}(r) & = & g \\ \text{etc.} & & & & & \end{array}$$

The subtype hierarchy is given by:

$$\begin{array}{lll} X_2 & \mathsf{Sub} & X_3 \\ X_4 & \mathsf{Sub} & X_3 \end{array}$$

2.5 The Object Relation Network

The functional drawing style (see figure 5) presents object types as the nodes of a labelled graph. For the Object Relation Network, we restrict ourselves (in case of atomic objects) to those that are pater familias ($\sqcap(x) = x$). The predicators are the labelled edges of this graph: there is an edge from x (or $\sqcap(x)$ if x atomic) to y with label p, if $p \in \mathcal{P}$, $\mathsf{Base}(p) = x$ and $\mathsf{Fact}(p) = y$. The resulting network is called the *Object Relation Network*. We call an information structure cyclic if the associated Object Relation Network contains a (directed) cycle.

Example 2.3 *The Object Relation Network corresponding to the information structure from figure 1 is drawn in figure 5.*

Example 2.4 *The Object Relation Network, corresponding to figure 8 is given in figure 9.*

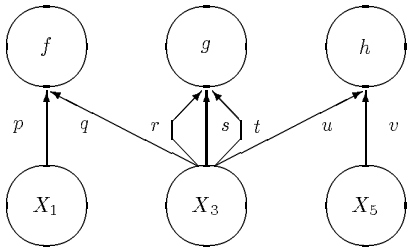


Figure 9: Example information structure as Object Relation Network

2.6 Populations

An information structure is used as a frame for some part of the (real) world, the so-called Universe of Discourse (UoD). A *state* of the UoD then corresponds with a so-called instantiation or population of the information structure, and vice versa. The idea of states was previously mentioned in [8], [26], [22]. Furthermore, a state transition of the UoD has a corresponding transition on populations of the information structure. This can be formulated as:

> The Universe of Discourse is *isomorphic* with the set of possible populations of the information structure and a transition relation hereupon.

This is called the *conceptuality property* of information structures.

In our approach, a population Pop of an information structure $\mathcal{I} = \langle \mathcal{P}, \mathcal{O}, \mathsf{Sub}, \mathcal{F} \rangle$ is a value assignment to the object types in $\mathcal{O}$, conforming to the structure as prescribed in $\mathcal{P}$ and $\mathcal{F}$, respecting the subtype hierarchy Sub. This is denoted as $\mathsf{IsPop}(\mathcal{I}, \mathsf{Pop})$. In the sequel of this paper we will omit quantifications over Pop.

Respecting the subtype hierarchy is reflected by the *Subtype Rule*:

$$\forall_{x,y \in \mathcal{A}} \left[x \,\mathsf{Sub}\, y \Rightarrow \mathsf{Pop}(x) \subseteq \mathsf{Pop}(y) \right]$$

The population of an atomic object type is just a set of values. The population of a composed object type is a set of tuples. A tuple t of a fact type f is a mapping of all its predicators to values of the appropriate type. This is referred to as the *Conformity Rule*:

$$\forall_{f \in \mathcal{F}} \forall_{t \in \mathsf{Pop}(f)} \forall_{p \in f} \left[t(p) \in \mathsf{Pop}(\mathsf{Base}(p)) \right]$$

This latter property can be extended with the notion of connected populations, defined in the *Connectivity Rule*. This rule requires that atomic object occurrences can only exist by the virtue of having properties.

$$\mathsf{Connected}(\mathsf{Pop}) \equiv \forall_{a \in \mathcal{A}, \sqcap(a)=a} \forall_{x \in \mathsf{Pop}(a)} \exists_{p \in \mathcal{P}} \exists_{t \in \mathsf{Pop}(\mathsf{Fact}(p))} \left[t(p) = x \right]$$

This rule is typical for NIAM, but is not required in ER.

Usually atomic values within populations are required to be strongly typed. This is formulated as follows:

$$\forall_{x,y \in \mathcal{A}} \left[\mathsf{Pop}(x) \cap \mathsf{Pop}(y) \neq \emptyset \Rightarrow \sqcap(x) = \sqcap(y) \right]$$

This rule is referred to as the *Strong Typing Rule*.

3 Derived Fact Types

The relational algebra has been introduced as a retrieval language for the relational model. We introduce this algebra also for the Predicator Model, as a mechanism to describe so-called derived relation types (derived fact types). Derived fact types are described by an algebraic expression, from which the type of the fact type can be derived, and that gives a rule for calculating its value (instantiation) in any population of the information structure. Using this basic language, more sophisticated retrieval languages can be described, for example languages based on path expressions (such as RIDL, see [24] or [27]).

The relational algebra $\boldsymbol{\mathcal{R}}(\mathcal{I})$, associated with an information structure $\mathcal{I}$, consists of the set of relational expressions on $\mathcal{I}$. This set is introduced by an inductive definition. This gives us the opportunity to use structural induction over $\boldsymbol{\mathcal{R}}(\mathcal{I})$.

3.1 Relational Expressions

A relational expression (derived fact type) r is either a fact type or a relational operator applied to one (or more) expression(s). If it is a fact type, say f, the schema $\mathsf{Schema}(r)$ of this expression is the set of predicators in f ($\mathsf{Schema}(f) = f$). Otherwise, the schema can be derived from the fact types contained in it. The value of expression r can be calculated by the operator Pop, which yields a set of tuples over $\mathsf{Schema}(r)$.

The basic relational operators are union, difference, join, projection, selection, extension and unnesting. The expressions that can be derived from the base relations by these operators form the relational algebra associated with the actual information structure $\mathcal{I}$, and is denoted as $\boldsymbol{\mathcal{R}}(\mathcal{I})$. We start with union and difference. Suppose r and s are compatible relational expressions, meaning $\mathsf{Schema}(r) = \mathsf{Schema}(s)$. Then $r \cup s$ and $r \setminus s$ both are relational expressions, having schema $\mathsf{Schema}(r)$, and having the following value respectively:

- $\mathsf{Pop}(r \cup s) = \mathsf{Pop}(r) \cup \mathsf{Pop}(s)$

- $\mathsf{Pop}(r \setminus s) = \mathsf{Pop}(r) \setminus \mathsf{Pop}(s)$

If r and s are relational expressions, then the join $r \bowtie s$ is a relational expression, identified by:

1. $\mathsf{Schema}(r \bowtie s) = \mathsf{Schema}(r) \cup \mathsf{Schema}(s)$
2. $\mathsf{Pop}(r \bowtie s) = \{\, t \mid t[r] \in \mathsf{Pop}(r) \wedge t[s] \in \mathsf{Pop}(s) \,\}$

Suppose r is a relational expression, $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ are predicators, all $q_1, \ldots, q_n$ different, such that $p_i \sim q_i$ for $1 \leq i \leq n$, then $\pi_{q_1:p_1,\ldots,q_n:p_n}\, r$ is a relational expression, identified by:

1. $\mathsf{Schema}(\pi_{q_1:p_1,\ldots,q_n:p_n}\, r) = \{q_1, \ldots, q_n\}$
2. $\mathsf{Pop}(\pi_{q_1:p_1,\ldots,q_n:p_n}\, r) = \left\{\, t \;\middle|\; \exists_{s \in \mathsf{Pop}(r)} \forall_{1 \leq i \leq n} [t(q_i) = s(p_i)] \,\right\}$

This definition is in style with [18]. Note that the projection can be used to "rename" a predicator. Furthermore, the projection operator can be used to extend relations with new predicators. This is especially useful, when more sophisticated expressions are possible, to describe the value in a "new" predicator. We will use $\pi_{p_1,\ldots,p_n}\, r$ as a shorthand for $\pi_{p_1:p_1,\ldots,p_n:p_n}\, r$. If τ is a set of predicators, then the notation $\pi_\tau\, r$ is also used.

If r is a relational expression, and F a selection formula, then $\sigma_F\, r$ is also a relational expression, according to:

1. $\mathsf{Schema}(\sigma_F\, r) = \mathsf{Schema}(r)$
2. $\mathsf{Pop}(\sigma_F\, r) = \{\, t \in \mathsf{Pop}(r) \mid F(t) \,\}$

The extension operator χ extends a relational expression r with a new predicator a, counting the number of tuples with an equal τ-value, in the following way:

1. $\mathsf{Schema}(\chi(r, \tau, a)) = \mathsf{Schema}(r) \cup \{a\}$
2. $\mathsf{Pop}(\chi(r, \tau, a))$
   $= \left\{\, t \;\middle|\; t[\mathsf{Schema}(r)] \in \mathsf{Pop}(r) \wedge t(a) = \#_{s \in \mathsf{Pop}(r)}(t[\tau] = s[\tau]) \,\right\}$

This operator is a restricted version of the extension operator introduced in [2], which was introduced to handle GROUP BY queries in SQL.

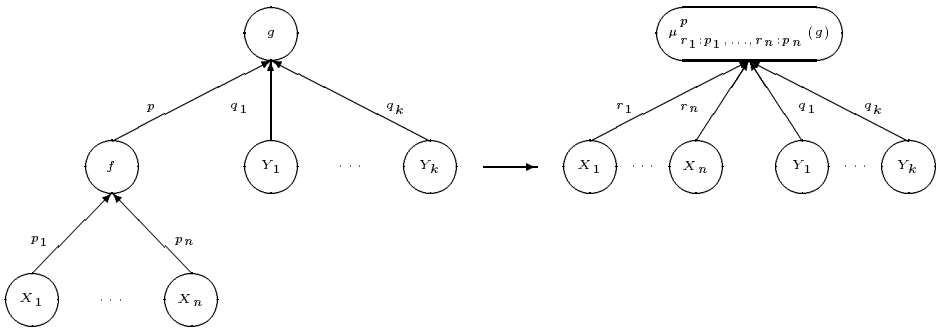


Figure 10: The functional look at unnesting an objectified relation

The unnest operator is used to flatten objectified relation types. This operator is defined for the NF^2 data model in [18]. In the predicator model, we need an alternative definition. Let f, g be relational expressions, and $p \in \mathsf{Schema}(g)$ a predicator such that $\mathsf{Base}(p) = \mathsf{Schema}(f)$ (see figure 10). Let $\mathsf{Schema}(f) = \{p_1, \ldots, p_n\}$. Furthermore, $r_1, \ldots, r_n$ are predicators not occurring in $\mathsf{Schema}(f)$ or $\mathsf{Schema}(g)$. Let $S = \mathsf{Schema}(g) \setminus \{p\}$. Then $\mu^p_{r_1:p_1,\ldots,r_n:p_n}(g)$ is a relational expression, defined by:

1. $\mathsf{Schema}(\mu^{p}_{r_1:p_1,\ldots,r_n:p_n}(g)) = \{r_1,\ldots,r_n\} \cup S$
2. $\mathsf{Pop}(\mu^{p}_{r_1:p_1,\ldots,r_n:p_n}(g)) = \{\, t \cup s[S] \mid t \in \mathsf{Pop}(f) \wedge s \in \mathsf{Pop}(g) \wedge s(p) = t \,\}$

In this definition the expression $t \cup s[S]$ relies on the introduction of a tuple as a mapping, while a mapping is mathematically defined as a set of pairs.

3.2 Boolean operations

In this section we look at boolean expressions. We start with the test IsEmpty to find out whether a relational expression has an empty population:

$$\mathsf{IsEmpty}(r) \equiv \mathsf{Pop}(r) = \emptyset$$

A very important notion in relational algebra theory is functional dependency. Let r be a relational expression and let $\sigma, \tau \subseteq \mathsf{Schema}(r)$. Then $\sigma \xrightarrow{r} \tau$ is a boolean expression which holds in a population Pop iff in this population, $\mathsf{Pop}(r)$ can be seen as a mapping from $\mathsf{Pop}(\pi_\sigma(r))$ into $\mathsf{Pop}(\pi_\tau(r))$ in the obvious way. We then call τ functionally dependent on σ via r.

Comparing relational expressions is only useful in case of relational expressions with different schema's, but particularly when the bases of the predicators can be matched to each other. This is especially of importance for set relations.

We start with the subset operator. Let r and s be relational expressions, then we call ϕ a *match* between $\mathsf{Schema}(r)$ and $\mathsf{Schema}(s)$ if ϕ is a bijection between $\mathsf{Schema}(r)$ and $\mathsf{Schema}(s)$, such that $\forall_{p \in \mathsf{Schema}(r)}\,[p \sim \phi(p)]$. Then $r \subseteq_\phi s$ is a boolean expression according to:

$$\forall_{t \in \mathsf{Pop}(r)} \exists_{u \in \mathsf{Pop}(s)} \forall_{p \in \mathsf{Schema}(r)}\,[t(p) = u(\phi(p))]$$

The equality operator can be defined in terms of subset relations:

$$r =_\phi s \Leftrightarrow r \subseteq_\phi s \wedge s \subseteq_{\phi^{-1}} r$$

The exclusion test $\otimes_\phi$ is defined as:

$$\forall_{t \in \mathsf{Pop}(r)} \neg \exists_{u \in \mathsf{Pop}(s)} \forall_{p \in \mathsf{Schema}(r)}\,[t(p) = u(\phi(p))]$$

3.3 Special operations

The count operator is used for counting the number of tuples in a relational expression. If r is a relational expression, then $\mathbb{N}(r)$ is defined by:

$$\mathbb{N}(r) = |\mathsf{Pop}(r)|$$

We can also count the number of different values on a set τ of predicators:

$$\mathbb{N}(r, \tau) = \mathbb{N}(\pi_\tau\, r)$$

If the base of a predicator is an ordered set, then we can ask the extreme values for this predicator by:

$$\begin{aligned} \mathsf{min}(r, p) &= \mathsf{min}(\mathsf{Pop}(\pi_p\, r)) \\ \mathsf{max}(r, p) &= \mathsf{max}(\mathsf{Pop}(\pi_p\, r)) \end{aligned}$$

The operator Range_p coerces a unary relation $f = \{p\}$ for some predicator p into the range of values that are taken by this single predicator:

$$\mathsf{Pop}(\mathsf{Range}_p(r)) = \left\{ x \;\middle|\; \exists_{t \in \mathsf{Pop}(r)}\,[t(p) = x] \right\}$$

Usually, this coercion is obvious from the context, and therefore omitted.

4 Constraints

Usually not all possible populations of an information structure are valid, i.e. correspond with some state in the UoD. Forbidden populations are excluded by so-called static constraints. We will discuss total role, uniqueness, occurrence frequency, exclusion, equality, subset and enumeration constraints. The semantics of constraints will be expressed in terms of relational expressions. For related work, see [14].

Dynamic constraints exclude forbidden transitions. They are not considered in this paper. Several approaches exist for dynamic constraints, for example, temporal logic ([8]), predicate transition networks ([17], [10]) and algebraic specification ([1]).

A schema $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$ consists of an information structure $\mathcal{I}$ (the syntactical part) and a set of constraints $\mathcal{C}$ (the semantical part). A population of a schema should be syntactically correct, and satisfy the semantic requirements:

$$\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \equiv \mathsf{IsPop}(\mathcal{I}, \mathsf{Pop}) \wedge \forall_{c \in \mathcal{C}} [\mathsf{Pop} \models c]$$

In the next sections we describe what kind of constraints are considered. The set of all possible constraints then is denoted by $\boldsymbol{\Gamma}(\mathcal{I})$. Note that $\mathcal{C} \subseteq \boldsymbol{\Gamma}(\mathcal{I})$.

4.1 Total role constraint

An object type may have the property that in any population, all its instances must be involved in some set of predicators. This property is called a total role constraint.

Let τ be a nonempty subset of $\mathcal{P}$. A total role constraint τ makes only sense if all predicators in τ have the same pater familias:

$$\forall_{p,q \in \tau} [p \sim q]$$

A population Pop satisfies the total role constraint $\mathsf{total}(\tau)$, denoted as $\mathsf{Pop} \models \mathsf{total}(\tau)$, if it satisfies the following expression:

$$\bigcup_{q \in \tau} \mathsf{Base}(q) = \bigcup_{q \in \tau} \pi_q(\mathsf{Fact}(q))$$

Note the implicit use of the coercion operator (see section 3.3). This condition is checked by (after some simplifications, based on the properties of the union-operator):

$$\bigcup_{q \in \tau} \mathsf{Pop}(\mathsf{Base}(q)) = \bigcup_{q \in \tau} \mathsf{Pop}(\pi_q(\mathsf{Fact}(q)))$$

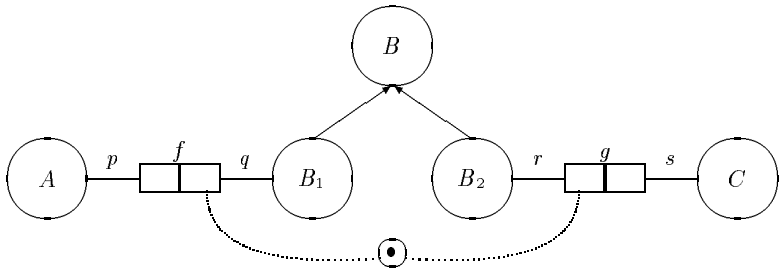


Figure 11: Example of total role constraint

Example 4.1 *Consider the total role constraint* $\mathsf{total}(\{q, r\})$ *in the information structure of figure 11. The meaning of this constraint is:*

$$B_1 \cup B_2 = \pi_q f \cup \pi_r g$$

Connectedness of a population can be expressed as the conjunction of total role constraints for atomic objects:

Lemma 4.1

$$\mathsf{Connected}(\mathsf{Pop}) \Leftrightarrow \forall_{a \in \mathcal{A}, \sqcap(a)=a} \left[\mathsf{Pop} \models \mathsf{total}(\{\ p \in \mathcal{P}\ \mid \mathsf{Base}(p) = a\ \})\right]$$

Proof: Let $B_a = \{\ p \in \mathcal{P} \mid \mathsf{Base}(p) = a\ \}$, then we can rewrite the righthand side as:

$$\forall_{a \in \mathcal{A}, \sqcap(a)=a} \left[\mathsf{Pop}(a) = \bigcup_{q \in B_a} \mathsf{Pop}(\pi_q\, \mathsf{Fact}(q))\right]$$

Now the result easily follows.

□

4.2 Uniqueness constraint

The uniqueness of values in some set of predicators has become a widely used concept in database technology, for guaranteeing integrity and as a base for efficient access mechanisms. In this section we will define some well-known concepts, such as functional dependency, candidate keys and uniqueness constraints.

Let r be a relational expression and let $\sigma, \tau \subseteq \mathsf{Schema}(r)$. Then, in population Pop, we call τ functionally dependent on σ over r, if relation $\sigma \xrightarrow{r} \tau$ holds in population Pop:

$$\mathsf{Pop} \models \sigma \xrightarrow{r} \tau$$

The following properties are obvious:

Lemma 4.2

$$\sigma \xrightarrow{r} \tau \Leftrightarrow \max(\chi(r, \sigma, a), a) = 1$$

Lemma 4.3

$$\mathsf{Schema}(r) \xrightarrow{r} \mathsf{Schema}(r)$$

A set of predicators $\sigma \subseteq \mathsf{Schema}(r)$ is called a candidate key of relational expression r if $\sigma \xrightarrow{r} \mathsf{Schema}(r)$ and σ is minimal for this property:

$$\forall_\tau \left[\tau \xrightarrow{r} \mathsf{Schema}(r) \Rightarrow \tau \not\subset \sigma\right]$$

A candidate key is also referred to as an identifier ([15]). We use $\mathsf{identifier}(r, \sigma)$ as an expression to denote that σ is an identifier of r.

Lemma 4.4

$$\forall_{f \in \mathcal{F}} \exists_{\sigma \subseteq f} \left[\mathsf{Pop} \models \mathsf{identifier}(f, \sigma)\right]$$

Proof: From lemma 4.3 we know $\mathsf{Schema}(f) \xrightarrow{f} \mathsf{Schema}(f)$. Then either $\mathsf{Schema}(f)$ is a candidate key, or some subset $K \subseteq \mathsf{Schema}(f)$ has the property $K \xrightarrow{f} \mathsf{Schema}(f)$, in which case the argument is recursively applied. The finiteness of $\mathcal{P}$ (see section 2.2) guarantees termination of this argument.

□

Remark: Using structural induction on the construction of relational expressions, this property is easily extended to all relational expressions.

A uniqueness constraint $\bigcup(\sigma)$ is a non-empty set of predicators $\sigma \subseteq \mathcal{P}$. The semantics of this constraint will be expressed as an identifier of the form $\mathsf{identifier}(\xi(\sigma), \sigma)$. The operator ξ will be introduced in the rest of this section.

4.2.1 Single fact type

When σ does not exceed the boundaries of a single fact type f (i.e. $\sigma \subseteq f$), the uniqueness constraint is bound to this relation: $\xi(\sigma) = f$.

4.2.2 Joinable via common object types

When more than one fact type is involved, these fact types may be *joinable via common object types*:

$$\mathsf{Jn}(\sigma) \equiv |\mathsf{Facts}(\sigma)| > 1 \Rightarrow \exists_{p \in \sigma} \left[\begin{array}{c} \exists_{q \in \sigma \setminus \mathsf{Fact}(p)} [\mathsf{Fact}(p) \setminus \sigma \sim \mathsf{Fact}(q) \setminus \sigma] \\ \wedge\, \mathsf{Jn}(\sigma \setminus \mathsf{Fact}(p)) \end{array} \right]$$

Two sets of predicators τ_1 and τ_2 are joinable via common object types ($\tau_1 \sim \tau_2$) if $\exists_{p \in \tau_1} \exists_{q \in \tau_2} [p \sim q]$. A direct consequence is that in this type of uniqueness constraint, no entire fact type can be involved. The uniqueness constraint σ then specifies a candidate key for the derived relation:

$$\xi(\sigma) = \sigma_{C(\sigma)} \bowtie_{f \in \mathsf{Facts}(\sigma)} f$$

where $\mathsf{Facts}(\sigma)$ is the set of fact types that are involved in the uniqueness constraint: $\{\, \mathsf{Fact}(p) \mid p \in \sigma \,\}$. Let $\tau(\sigma) = \bigcup \mathsf{Facts}(\sigma) \setminus \sigma$ be the set of predicators that are not involved in the uniqueness condition. The selection condition then is defined as follows:

$$C(\sigma) = \bigwedge_{p,q \in \tau(\sigma),\, p \sim q} p = q$$

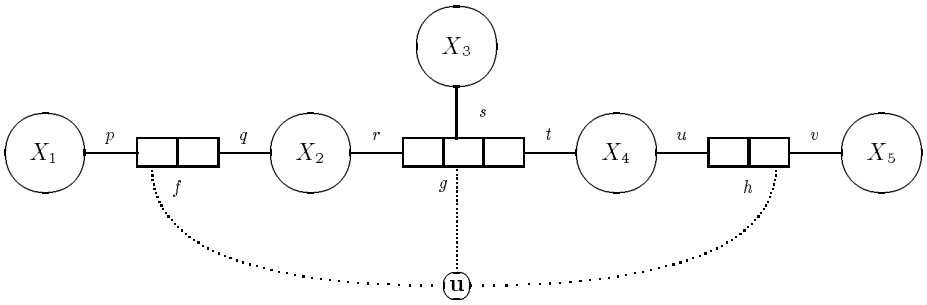


Figure 12: An example uniqueness constraint

Example 4.2 *In figure 12 uniqueness constraint $\sigma = \{p, s, v\}$ is specified. Consequently $\mathsf{Facts}(\sigma) = \{f, g, h\}$, $\tau(\sigma) = \{q, r, t, u\}$ and thus $C(\sigma) = (q = r \wedge t = u)$. The condition $\mathcal{U}(\sigma)$ requires that σ is a candidate key of*

$$\xi(\sigma) = \sigma_{q=r \wedge t=u}(f \bowtie g \bowtie h)$$

The condition 'joinable via common object types' is easily interpreted in the Object Relation Network (see figure 13) as an (undirected) path connecting all fact types involved.

Example 4.3 *In figure 14 we have a peculiar uniqueness constraint: $\alpha = \{r, t\}$. Our interpretation leads to $\tau(\alpha) = \{p, q, s\}$, and consequently:*

$$\mathsf{identifier}(\sigma_{p=s \wedge q=s}(f \bowtie g), \{r, t\})$$

From this example, it is clear that the graphical notation as used in NIAM, is not powerful enough to allow for alternative join conditions, e.g. $q = s$. This problem can be solved by the specification of a subset $\tau(\sigma)$ of predicators that are to be joined.

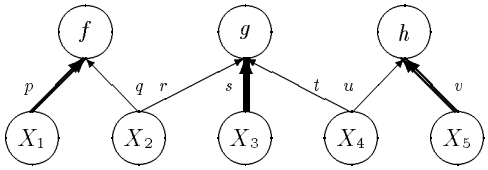


Figure 13: The same uniqueness constraint drawn functionally

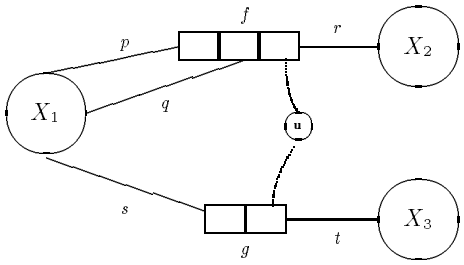


Figure 14: Another uniqueness constraint

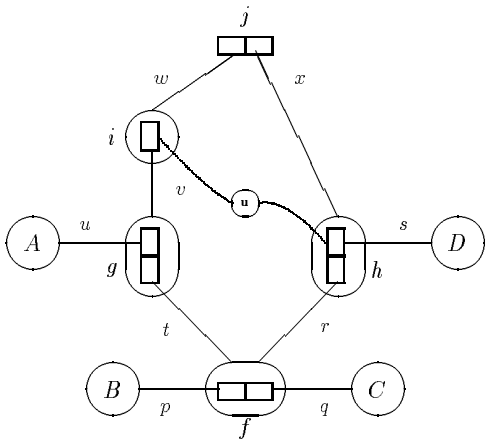


Figure 15: A uniqueness constraint over objectification

4.2.3 Uniqueness and Objectification

Next we consider uniqueness constraints in relation with objectification. In figure 15 we see an example of a rather bizar nature. The functional representation is found in figure 16. In this example, $\bigcup(\{u, s\})$ is a joinable uniqueness constraint. This is not the case for $\sigma = \{v, s\}$. In order to make $\mathsf{Facts}(\sigma)$ joinable, we look for the *highest common descendants* of the involved fact types $\mathsf{Fact}(v) = i$ and $\mathsf{Fact}(s) = h$, and find f as the bridge between these fact types (*joinable via common descendants*). We then flatten (unnest) the ancestor fact types, until they have f as an immediate descendant, and arrive at the premises of the previous section.

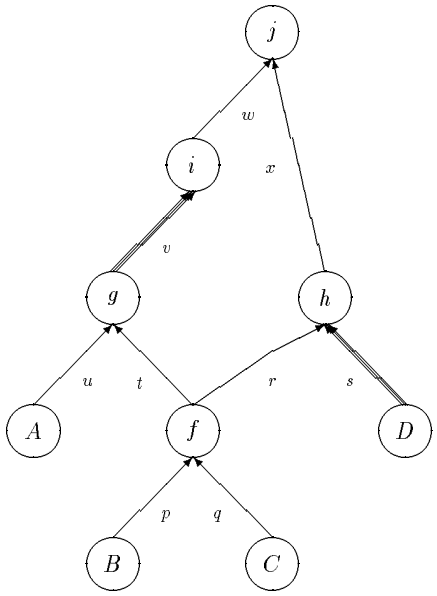


Figure 16: The functional view

4.2.4 The Uniquest Algorithm

The general interpretation of a uniqueness constraint can now be formulated, and is called the *Uniquest Algorithm*. Consider $\bigcup(\sigma)$ for $\sigma \neq \emptyset$. Let $G(\sigma)$ be the subgraph of the Object Relation Network that is relevant for σ, or, the minimal connected subgraph containing σ. Then $\xi(\sigma)$ is computed as $\xi(\sigma, G(\sigma))$, where:

$\xi(\sigma, G) =$
if all edges in σ have same destination f
then return f
elif some top f with heigth > 1 in G has incoming edge $p \notin \sigma$
then return $\xi\left(\sigma, G\left[f := \mu^{p}_{L}(f)\right]\right)$
for some proper rename list L
elif all tops have height 1, and connected via common object types
then return $\sigma_{C(\sigma)}\left(\bowtie_{f \in \mathsf{Facts}(\sigma)} f\right)$
else error: no joinable descendants
fi

The rename list L should be such that renamings are only performed when required. A renaming of predicators is only required when the top f has more predicators between $\mathsf{Base}(p)$ and $\mathsf{Fact}(p)$.

Example 4.4 *The uniqueness constraint* $\bigcup(\{p, s, u\})$ *in the information structure of figure 17 has the following semantics:*

$$\mathsf{identifier}\left(\mu^{r}_{p:p,q:q}(g) \bowtie f \bowtie \mu^{t}_{p:p,q:q}(h), \{p, s, u\}\right)$$

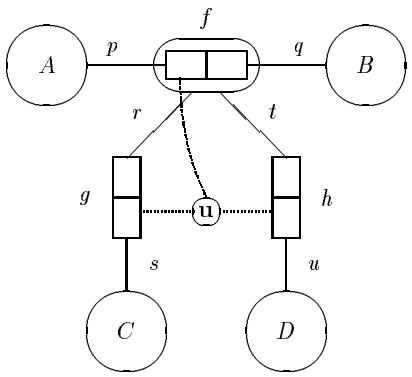


Figure 17: The information structure

Example 4.5 *Applying the Uniquest Algorithm to the situation of figure 16 yields the error: no joinable descendants.*

For an extended set of examples of the behaviour of the Uniquest Algorithm, see [29].

4.3 Occurrence frequency constraint

Uniqueness constraints are used to express that instances of object types may play a certain combination of roles at most once. This restriction can be generalised as follows: if a certain combination of object type instances occurs in a set σ of predicators, then this combination should occur at least n and at most m times in this set. This is denoted as:

$$\mathsf{frequency}(\sigma, n, m)$$

The semantics of this expression are:

$$\neg\, \mathsf{IsEmpty}(\xi(\sigma)) \Rightarrow \begin{cases} \min\left(\chi(\xi(\sigma), \sigma, a), a\right) & \geq & n \\ \max\left(\chi(\xi(\sigma), \sigma, a), a\right) & \leq & m \end{cases}$$

Of course we have:

$$\begin{aligned} \cup(\sigma) &\Leftrightarrow \mathsf{frequency}(\sigma, 0, 1) \\ &\Leftrightarrow \mathsf{frequency}(\sigma, 1, 1) \end{aligned}$$

Note that the constraint $\mathsf{frequency}(\sigma, 0, 0)$ is a very strong condition, as it constrains $\xi(\sigma)$ to the empty population. As a result, it can be used to exclude unwanted situations.

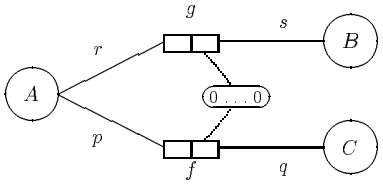


Figure 18: Example of an occurrence frequency constraint

Example 4.6 *The occurrence frequency constraint* $\mathsf{frequency}(\{q, s\}, 0, 0)$ *in figure 18 enforces:*

$$\pi_p f \cap \pi_r g = \emptyset$$

We will come back to this example in the next section.

4.4 Set constraints

Another type of frequently occurring constraints are so-called set constraints. Let σ and τ sets of predicators, and ϕ a match between σ and τ. Then the constraints $\otimes_\phi(\sigma,\tau)$, $\mathsf{equal}_\phi(\sigma,\tau)$, $\mathsf{exclusion}_\phi(\sigma,\tau)$ have the following interpretation respectively:

1. $\pi_\sigma(\xi(\sigma)) \subseteq_\phi \pi_\tau(\xi(\tau))$
2. $\pi_\sigma(\xi(\sigma)) =_\phi \pi_\tau(\xi(\tau))$
3. $\pi_\sigma(\xi(\sigma)) \otimes_\phi \pi_\tau(\xi(\tau))$

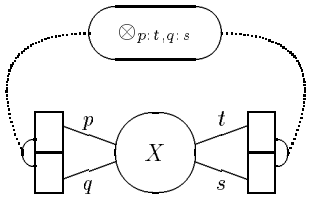


Figure 19: An example exclusion constraint

Usually, the bijection ϕ is immediate from the context, and is omitted. In figure 19 however, we see an example where ϕ is not clear from the context.

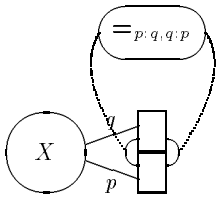


Figure 20: A symmetric homogeneous relation

The power of our approach is demonstrated best by the elegant way in which a symmetric homogeneous relation can be specified (see figure 20).

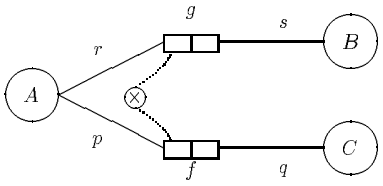


Figure 21: Occurrence frequency constraint as exclusion constraint

Example 4.7 *The occurrence frequency constraint of figure 18 is specified as an exclusion constraint in figure 21.*

4.5 Enumeration constraint

An enumeration constraint is used to bind a label type to an enumerated domain. If l is a label type, and V a set of values, then the constraint $\mathsf{enumeration}(l,V)$ requires:

$$\mathsf{Pop}(l) \subseteq V$$

4.6 Subtyping

Let $\phi \subseteq \mathcal{E}$ be a family of entity types, i.e.:

$$\forall_{x,y \in \phi} [\sqcap(x) = \sqcap(y)]$$

The lowest common ancestor of ϕ is denoted as $\sqcap(\phi)$.

A population Pop satisfies the exclusion subtype constraint $\mathsf{exclusion}(\phi)$, denoted as $\mathsf{Pop} \models \mathsf{exclusion}(\phi)$, iff:

$$\forall_{x,y \in \phi} [\mathsf{Pop}(x) \cap \mathsf{Pop}(y) = \emptyset]$$

A population Pop satisfies the total subtype constraint $\mathsf{total}(\phi)$, denoted as $\mathsf{Pop} \models \mathsf{total}(\phi)$, iff:

$$\mathsf{Pop}(\sqcap(\phi)) = \bigcup_{q \in \phi} \mathsf{Pop}(q)$$

4.7 Subtype Defining Rules

A subtype defining rule is a constraint $\mathsf{SubRule}(s, t, r)$, where s and t are object types, such that $s \,\mathsf{Sub}\, t$, and r a relational expression, having a singleton schema $\{p\}$ with $\mathsf{Base}(p) = t$. The meaning of this rule is:

$$s = r$$

(using the implicit coercion from section 3.3). A subtype defining rule is intended as a criterion for deciding whether an element of a (super)type t also belongs to the subtype s. As a result, we need a subtype defining rule for each subtype. More than one subtype defining rule is not allowed, as this may lead to contradictions. We denote the unique subtype defining rule for subtype t as $\mathsf{SubRule}(t)$.

As a consequence, the subtype defining rule should not depend on any subtype of s. Furthermore, cyclic subtype defining rules are also not possible. We formalise this by introducing $\mathsf{Facts}(r)$ as the set of fact types needed to evaluate r. An inductive definition of this operator is easily done, and is omitted here.

The condition that a subtype cannot be defined in terms of its own subtypes (if any) can now be formulated as:

$$\forall_{f \in \mathsf{Facts}(r)} \forall_{p \in f} [\neg \,\mathsf{Base}(p) \,\mathsf{Sub}\, s]$$

The requirement that cyclic subtype defining rules are not allowed is handled by the introduction of a relation Uses over entities:

$$a \,\mathsf{Uses}\, b \Leftrightarrow b \in \mathsf{Facts}(\mathsf{SubRule}(a))$$

The requirement then states that the resulting network should not contain a (directed) cycle.

Example 4.8 *Consider the subtype hierarchy in figure 11. The latter rule then excludes the following set of subtype defining rules:*

- $\mathsf{SubRule}(B_1, B, \pi_{r\ g})$
- $\mathsf{SubRule}(B_2, B, \pi_{q\ f})$

5 Identification

We discuss the identification for each class of object types. First we consider identification at the instance level. We call this *weak identification*. Then we consider *structural identification*.

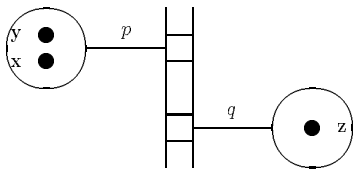


Figure 22: x and y both related to z

5.1 Weak identification

Labels are the elementary data types, and are considered to be representable directly. As a result, each label can be identified by itself. Also we assume an equality operator for labels. This operator decides on the equality of label values.

Entities on the other hand can only be represented by their properties. As a consequence, entities with the same properties are not distinguishable, and therefore considered to be the same. The properties of an entity are recorded by the facts in which they play a role. As a consequence, in any population Pop we have for all entity values x and y (see also figure 22):

$$\forall_{p \in \mathcal{P}, x,y \in \mathsf{Pop}(\mathsf{Base}(p))} \forall_{q \in \mathsf{Fact}(p)} [\mathsf{Identical}(x,y,p,q)] \Rightarrow x = y$$

where $\mathsf{Identical}(x,y,p,q)$ is defined as:

$$\begin{aligned} &\mathsf{Identical}(x,y,p,q) \\ &\quad \equiv \quad \forall_{z \in \mathsf{Pop}(\mathsf{Base}(q))} \left[\mathsf{Related}_{\mathsf{Pop}}(x,p,q,z) \Leftrightarrow \mathsf{Related}_{\mathsf{Pop}}(y,p,q,z) \right] \end{aligned}$$

and $\mathsf{Related}_{\mathsf{Pop}}$ is defined by:

$$\mathsf{Related}_{\mathsf{Pop}}(x,p,q,z) \equiv \exists_{t \in \mathsf{Pop}(\mathsf{Fact}(p))} [t(p) = x \wedge t(q) = z]$$

This rule is referred to as the *Weak Identification Rule*, and is denoted as $\mathsf{WeakId}(\mathcal{I}, \mathsf{Pop})$. The rule is typical for systems that deal only with complete knowledge.

Composite objects were introduced as tuples, and are identified by their components. As a consequence, in any population Pop we have for all values s and t of any fact type f:

$$\forall_{p \in f} [s(p) = t(p)] \quad \Rightarrow \quad s = t$$

This is called the *Extensionality Rule*. Note that composite objects are considered only in the case of complete knowledge. The reason is that tuples are defined as (total) functions, rather than partial functions.

The Weak Identification Rule guarantees that no naming conflicts for objects can occur. However, the rule does not enforce that every object has a name. So, in general, weak identification does *not* guarantee connectedness.

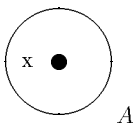


Figure 23: Population weakly identified

Example 5.1 *In figure 23 we see a very simple information structure, with a population that is weakly identified. Note that x is an anonymous object.*

Example 5.2 *In figure 24 this information structure has a population that is not weakly identified since it contains more than one anonymous object.*

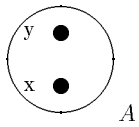


Figure 24: Population not weakly identified

5.2 Structural identification

Weak identification is an important property for information systems, as it ensures that all objects can be addressed uniquely. In this section we consider how this can be guaranteed from properties of the schema.

We call an information structure *structural identifiable* when there are no dangling objects:

1. Each label type occurs in some total role constraint:

$$\forall_{x \in \mathcal{L}} \exists_{p \in \mathcal{P}} \exists_{\mathsf{total}(\tau) \in \mathcal{C}} [\mathsf{Base}(p) = x \wedge p \in \tau]$$

   The motivation behind this is to enforce the absence of unused label values.

2. All entities can be identified:

$$\forall_{x \in \mathcal{E}} [\mathsf{Identifiable}(x)]$$

   The underlying idea is that entities that can not be distinguished from each other, are considered to be identical.

The predicate Identifiable is defined by:

1. If x is a label type, then obviously $\mathsf{Identifiable}(x)$.

2. If x is a composed object type (or, generally, a set of predicators), then we may conclude $\mathsf{Identifiable}(x)$ if all components of x are identifiable:

$$\forall_{p \in x} [\mathsf{Identifiable}(\mathsf{Base}(p))]$$

3. If x is an entity type, then we have the following cases:

   (a) If x is a subtype, i.e. $\sqcap(x) \neq x$, then x is identifiable if its associated pater familias is. Besides, there should be a unique subtype defining rule (see section 4.7).

   (b) In the other case $\sqcap(x) = x$. Then x can be identified, if there exists a set τ of predicators that can be used for this purpose, a so-called *identifier* for x, i.e.:

   - $\mathsf{Identifiable}(\tau)$
   - $\mathsf{U}(\tau)$
   - $\forall_{f \in \mathsf{Facts}(\tau)} \exists_{p \in \mathsf{Compl}(\tau, x) \cap f} [\mathsf{U}(p) \wedge \mathsf{total}(p)]$

The set $\mathsf{Compl}(\tau, x)$ of co-roles with respect to x is defined as:

$$\mathsf{Compl}(\tau, x) = \{\, p \in \bigcup \mathsf{Facts}(\tau) \setminus \tau \mid \mathsf{Base}(p) = x \,\}$$

Example 5.3 *Consider the schema of figure 25. Assume that A and B are identifiable. In order to identify X, we choose* $\tau = \{p\}$, *then* $\mathsf{Identifiable}(\tau)$ *and* $\mathsf{U}(\tau)$. *Furthermore* $\mathsf{Compl}(\tau, X) = \{q\}$, *while* $\mathsf{total}(q) \wedge \mathsf{U}(q)$.

In order to identify Y, we choose $\tau = \{t\}$, *then* $\mathsf{Identifiable}(\tau)$ *and* $\mathsf{U}(\tau)$. *Furthermore* $\mathsf{Compl}(\tau, Y) = \{r, s\}$, *while* $\mathsf{total}(r) \wedge \mathsf{U}(r)$.

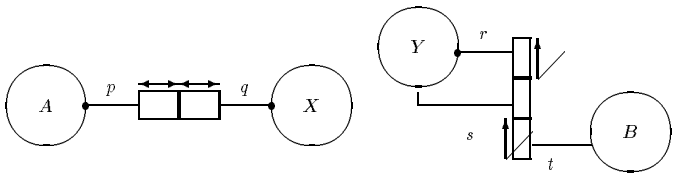


Figure 25: Simple identification

This definition of identifiability is rather strong: it does not allow cyclic information structures (as a result of the second rule). A direct consequence of the third rule is:

Corollary 5.1 *If Σ is identifiable, then it satisfies the* Atomic Anchorage Rule*:*

$$\forall_{x \in \mathcal{A}} \exists_{p \in \mathcal{P}} \, [\mathsf{Base}(p) = x]$$

For entity types this can be sharpened to:

$$\forall_{x \in \mathcal{E}} \exists_{p \in \mathcal{P}} \, [\mathsf{Base}(p) = x \wedge \mathsf{total}(\{p\}) \in \mathcal{C}]$$

For label types, it is sufficient that it is the base of a predicator, that occurs in some total role constraint.

Theorem 5.1 $\forall_{x \in \mathcal{E}} \, [\mathsf{Identifiable}(x)] \Rightarrow \forall_{x \in \mathcal{O}} \, [\mathsf{Identifiable}(x)]$

Proof: If all entity types of an information structure are identifiable, this structure is acyclic, and therefore we can define the depth of an object as the maximal distance in the Object Relation Network to an atomic object type. As a consequence, this distance is 0 for atomic objects. The statement is now easily proved by induction on the depth of objects.

□

An important consequence of identifiability is that it guarantees that each population will be weakly identified.

Theorem 5.2

$$\mathcal{I} \textit{ identifiable} \Rightarrow \textit{each population weakly identified}$$

Proof: Suppose $\mathcal{I}$ is identifiable. Now let e_1 and e_2 be occurrences of an object type x, having precisely the same properties. Let τ be an identifier for x, then e_1 and e_2 maintain via τ relations with the same objects. From the unicity of τ, we then conclude $e_1 = e_2$.

□

Another important consequence of identifiability is that it guarantees that each population will be connected.

Theorem 5.3

$$\mathsf{Identifiable}(\Sigma) \Rightarrow \forall_{\mathsf{Pop}} \, [\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \Rightarrow \mathsf{Connected}(\mathsf{Pop})]$$

Proof: Suppose $\mathsf{Identifiable}(\Sigma)$, and let Pop be a population of Σ. Now let $a \in \mathcal{A}$ with $\sqcap(a) = a$. From the Atomic Anchorage Rule we conclude that there exists a predicator p with $\mathsf{Base}(p) = a$, that is part of some total role constraint. From this, the result is easily derived.

□

Example 5.4 *In figure 26 we see an example of complex identification. The bases of predicators p_1, p_5 and p_9 are label types. Entity type* House *can be identified by identifier $\{p_7, p_9\}$, which recursively calls for the identification of* Street*. This can be done by identifier $\{p_3, p_5\}$, which recursively calls for the identification of* Community*.*

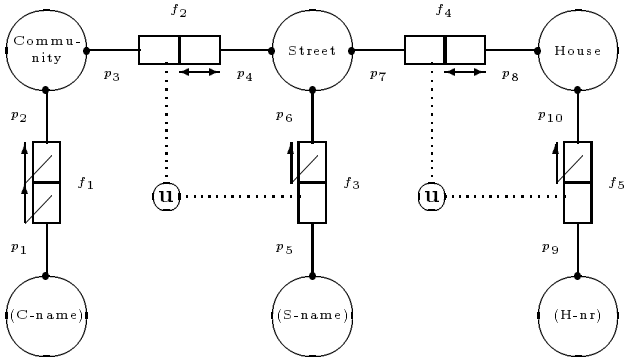


Figure 26: Complex identification

5.3 Cyclic Object Structures

In this section we consider cyclic information structures, i.e. information structures for which the associated Object Relation Network contains a directed cycle.

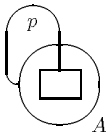


Figure 27: The elementary cyclic object

Example 5.5 *In figure 27 we see the most simple cyclic information structure, consisting of a single object, that is cyclic in itself: a predicator p with* $\mathsf{Base}(p) = \mathsf{Fact}(p)$. *Note that this (sub)schema is populatable, for example,* $\mathsf{Pop}(\mathsf{Base}(p)) = \{x\}$ *and* $\mathsf{Pop}(\mathsf{Fact}(p)) = \{x\}$.

In figure 28 we see another example of a cyclic information structure. This schema is also populatable, as can be easily verified.

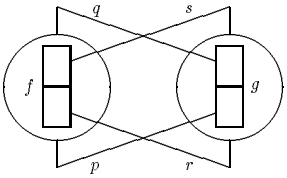


Figure 28: Two fact types contained in a cycle

The major problem with these cycles is, that structural identification is impossible (see section 5.2). Cycles can be easily detected by the topological sort algorithm (see [12]), applied on the Object Relation Network:

Theorem 5.4 *Let $G = \langle N, E \rangle$ be a directed graph, then G is acyclic iff there exists a function $h : N \rightarrow \mathbf{N}$, such that:*

$$\forall_{\langle s,d \rangle \in E} [h(s) < h(d)]$$

Example 5.6 *This theorem excludes the cycles in figure 27 and 28. We will show that a function h is impossible for the structure in figure 28. Let h be such a function. On the one hand* $\mathsf{Base}(p) = f$ *and thus $h(f) < h(g)$. On the other hand* $\mathsf{Base}(r) = g$, *which results in $h(g) < h(f)$, leading to a contradiction.*

6 Schema Properties

6.1 The Empty Population

Let schema $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$ be composed of an information structure $\mathcal{I}$ and a set of constraints $\mathcal{C}$. The empty population of Σ, is characterized by:

$$\mathsf{GlobEmpty}(\Sigma, \mathsf{Pop}) \equiv \mathsf{IsPop}(\mathcal{I}, \mathsf{Pop}) \wedge \forall_{x \in \mathcal{O}} [\mathsf{Pop}(x) = \emptyset]$$

Note that we only require Pop to be a syntactically correct population, as we can prove that the empty population is also semantically correct (i.e. all constraints are satisfied). We first show that the *law of the excluded miracle* holds for information systems, stating that we can not derive anything (any fact) when we have no information (no facts) at our disposal. This is a direct consequence of the closed world assumption.

Theorem 6.1 (law of the excluded miracle)

$$\mathsf{GlobEmpty}(\Sigma, \mathsf{Pop}) \Rightarrow \forall_{r \in \boldsymbol{\mathcal{R}}_{(\mathcal{I})}} [\mathsf{Pop}(r) = \emptyset]$$

Proof: Let Pop be an empty population of Σ. We use structural induction on the construction of relational expressions. Atomic relational expressions, being fact types, obviously have an empty population.

To prove the induction step, we suppose that r and s are relational expressions with an empty population. Then, the union, difference, projection, selection, extension and unnest operators applied to r and s obviously yield an empty population.

□

As a result, the empty population is semantically correct:

Theorem 6.2 $\mathsf{GlobEmpty}(\Sigma, \mathsf{Pop}) \Rightarrow \mathsf{IsPop}(\Sigma, \mathsf{Pop})$

Proof: Let Pop be an empty population of $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$. Then $\mathsf{IsPop}(\mathcal{I}, \mathsf{Pop})$. The constraints in $\mathcal{C}$ enforce certain relationships between (populations of) relational expressions. As all relational expressions on $\mathcal{I}$ have an empty population (lemma 6.1), all constraints are satisfied. This is easily checked for each type of constraint. As a result we have $\mathsf{IsPop}(\Sigma, \mathsf{Pop})$.

□

6.2 Non-empty Populations

Now, we focus on non-empty populations or, more specifically, on the question whether a schema allows non-emptiness, and to what extent. We introduce several forms of non-emptiness, and describe how they relate to each other. First, we ask ourselves whether each atomic object type can be populated at all. If so, the schema is called *local atomic populatable*:

$$\mathsf{LocAtomPop}(\Sigma) \equiv \forall_{a \in \mathcal{A}} \exists_{\mathsf{Pop}} [\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \wedge \mathsf{Pop}(a) \neq \emptyset]$$

For composed object types we define an analogous property. A schema is called *local fact populatable*, if each fact type can be populated:

$$\mathsf{LocFactPop}(\Sigma) \equiv \forall_{f \in \mathcal{F}} \exists_{\mathsf{Pop}} [\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \wedge \mathsf{Pop}(f) \neq \emptyset]$$

Populatability at the fact level is a stronger property than at the atomic level.

Lemma 6.1 $\mathsf{LocFactPop}(\Sigma) \Rightarrow \mathsf{LocAtomPop}(\Sigma)$

Proof: Let Σ be such that $\mathsf{LocFactPop}(\Sigma)$ and let $a \in \mathcal{A}$. If a is an isolated object type, then of course it can be populated. Otherwise a is the base of at least one predicator p. Since $\mathcal{F}$ is a partition of $\mathcal{P}$, p is involved in some fact type, say f. As Σ is local fact populatable, a population Pop exists, with $\mathsf{Pop}(f) \neq \emptyset$. Now, from the *Conformity Rule* we conclude that a is populated. Thus $\mathsf{LocAtomPop}(\Sigma)$.

□

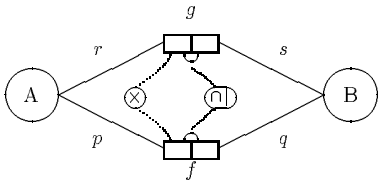


Figure 29: A non local fact populatable schema

Example 6.1 *The schema in figure 29 is not local fact populatable.*

Proof: We have $f = \{p, q\}, g = \{r, s\}$, with the following constraints:

- $\mathsf{exclusion}(\{p\}, \{r\})$.
- $\subseteq(\{r, s\}, \{p, q\})$.

Assume Σ is local fact populatable. Then there exists a population Pop, that populates g. As a consequence $\mathsf{Pop}(\pi_r g) \neq \emptyset$. From the subset constraint we derive $\mathsf{Pop}(\pi_r g) \subseteq \mathsf{Pop}(\pi_p f)$, and thus:

$$\mathsf{Pop}(\pi_r g) \cap \mathsf{Pop}(\pi_p f) = \mathsf{Pop}(\pi_r g) \neq \emptyset$$

This however contradicts the exclusion constraint. We conclude: Σ is not local fact populatable.

□

Having related both populatability properties to each other, we are ready to introduce two stronger properties of schemata. First we define *global atomic populatability*:

$$\mathsf{GlobAtomPop}(\Sigma) \equiv \exists_{\mathsf{Pop}} [\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \wedge \forall_{a \in \mathcal{A}} [\mathsf{Pop}(a) \neq \emptyset]]$$

The following lemma is trivial:

Lemma 6.2 $\mathsf{GlobAtomPop}(\Sigma) \Rightarrow \mathsf{LocAtomPop}(\Sigma)$

Global fact populatability is defined as:

$$\mathsf{GlobFactPop}(\Sigma) \equiv \exists_{\mathsf{Pop}} [\mathsf{IsPop}(\Sigma, \mathsf{Pop}) \wedge \forall_{f \in \mathcal{F}} [\mathsf{Pop}(f) \neq \emptyset]]$$

For this schema property we have:

Lemma 6.3 $\mathsf{GlobFactPop}(\Sigma) \Rightarrow \mathsf{GlobAtomPop}(\Sigma) \wedge \mathsf{LocFactPop}(\Sigma)$

Example 6.2 *In figure 30 we see a schema, that is local fact populatable and global atomic populatable, but not global fact populatable:*

Proof: *Assume* Pop *is a population of* Σ *such that each fact type has a non-empty population. The exclusion constraint between* p *and* r *implies that the population of* A *contains at least two elements, which contradicts the enumeration constraint on* A.

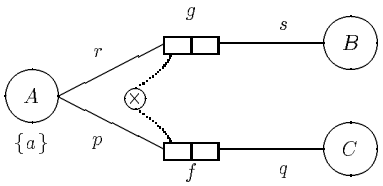


Figure 30: A non global fact populatable schema

□

The notion of *significant population* is an important concept during information analysis. According to [15], a population is called significant with respect to some kind of constraint if and only if all the relevant UoD constraints of that kind may be deduced from that population. We take a slightly different approach, and consider a population significant if it makes visible precisely all constraints of some kind $\mathcal{D} \subseteq \boldsymbol{\Gamma}(\mathcal{I})$. As a consequence, constraints of that kind that are not visible do not hold:

$$\begin{aligned} \mathsf{Signif}(\Sigma, \mathsf{Pop}, \mathcal{D}) \quad \equiv \quad & \mathsf{IsPop}(\Sigma, \mathsf{Pop}) \wedge \forall_{x \in \mathcal{O}} \left[\mathsf{Pop}(x) \neq \emptyset\right] \\ & \wedge \forall_{c \in \mathcal{D}} \left[\mathsf{Pop} \models c \Leftrightarrow \mathcal{C} \vdash c\right] \end{aligned}$$

For example, $\mathcal{D}$ could be the set of all total role constraints in $\boldsymbol{\Gamma}(\mathcal{I})$. We will sharpen this as follows: a schema is called significant populatable, denoted as $\mathsf{SignifPop}(\Sigma, \mathcal{D})$, if it can be populated with a significant population with respect to $\mathcal{D}$.

Lemma 6.4 $\mathsf{SignifPop}(\Sigma, \emptyset) \Rightarrow \mathsf{GlobFactPop}(\Sigma)$

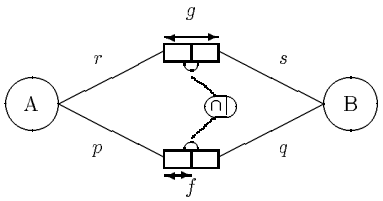


Figure 31: A non significant populatable schema

Example 6.3 *Figure 31 shows a schema which is easily seen to be global fact populatable. A close inspection however will reveal that this schema has a problem: the combination of uniqueness constraints does not harmonise with the subset constraint. This problem is captured as the schema is not significantly populatable with respect to uniqueness constraints.*

Proof: *Let* Pop *be a population such that each object type has a non-empty population. The uniqueness constraint over g implies at least the population for g as presented in table 1, with $a_1 \neq a_2$ and $b_1 \neq b_2$.*

r	s
a_1	b_1
a_1	b_2
a_2	b_1

Table 1: A significant population for g

The subset constraint $g \subseteq f$ now contradicts $b_1 \neq b_2$, by the uniqueness constraint over f.

□

7 Verification

In this section we consider the complexity of verifying schema properties. For schema $\Sigma = \langle \mathcal{I}, \mathcal{C} \rangle$ we take $|\mathcal{C}|$ as size of the problem. In order to prove the NP-completeness results, we make use of the so-called three dimensional matching problem.

7.1 The Three Dimensional Matching Problem

The three dimensional matching problem is a very popular problem for proving NP-completeness results. It is a generalisation of the well-known marriage problem. The matching problem $\mathcal{M}$ is formulated as follows ([9]):

> Let $M \subseteq W \times X \times Y$, where W, X and Y are disjoint sets having the same number of elements q. The problem is to determine whether M contains a so-called matching, i.e. a subset $M' \subseteq M$ with q elements, such that no two elements of M' agree in any coordinate.

This problem is known to be NP-complete (see [9]). Without loss of generality, we assume that every element of W, X and Y occurs in some element of M (which is easily checked). If this is not the case, no matching is possible.

Example 7.1 *Let* $W = \{w_1, w_2\}$, $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, *and* $M = \{(w_1, x_2, y_2), (w_2, x_1, y_1), (w_1, x_1, y_1), (w_2, x_2, y_2)\}$ *Then* M *contains several matchings, for example* $M' = \{(w_1, x_2, y_2), (w_2, x_1, y_1)\}$.

7.2 Global Atomic Populatability

In this section we consider the complexity of verifying the schema property of global atomic populatability. We start with the description of a transformation of a three dimensional matching problem $\mathcal{M}$ into the verification of the property of global atomic populatability of a schema $\Sigma(\mathcal{M})$.

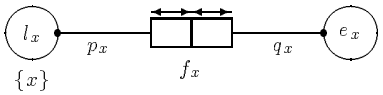


Figure 32: A fact type between l_x and e_x

For each element of $x \in W \cup X \cup Y$, we introduce a fact type $f_x = \{p_x, q_x\}$ (see figure 32), with $\mathsf{Base}(p_x) = l_x \in \mathcal{L}$ and $\mathsf{Base}(q_x) = e_x \in \mathcal{E}$.

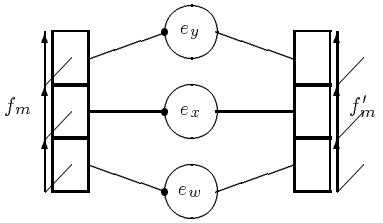


Figure 33: Two fact types for each $m \in M$

Each element $m \in M$ is transformed into two fact types. For $m = (w, x, y)$ we get the fact types f_m and f'_m, as shown in figure 33. Fact type f_m is intended to be populated with m. A non-empty population of f'_m will correspond to $m \in M'$.

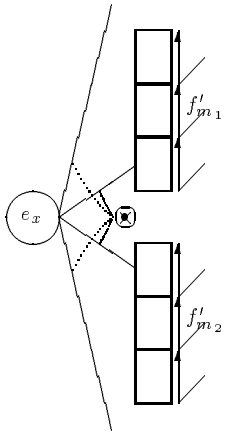


Figure 34: Matching conditions via constraints

The matching conditions (each element of W, X and Y should occur in the matching, and no two elements of M' may agree in any coordinate) are transformed into total role and exclusion constraints (respectively) for each element $x \in W \cup X \cup Y$ (see figure 34).

Obviously this tranformation takes not more than polynomial time. The correctness of this transformation is expressed by:

Theorem 7.1

$$\mathcal{M} \textit{ has a matching} \Leftrightarrow \mathsf{GlobAtomPop}(\Sigma(\mathcal{M}))$$

Proof:

$\Longrightarrow$ Let $M' \subseteq M$ be a matching. From M' we construct a population Pop, such that every atomic object type is populated. The population of fact type f_m will consist of the single fact m. If $m \in M'$, then f'_m will have the same population, otherwise it gets an empty population. It is easily verified that this population satifies all constraints. In this population obviously all atomic object types are populated.

$\Longleftarrow$ Assume $\mathsf{GlobAtomPop}(\Sigma(\mathcal{M}))$ and let Pop be such a global population. A matching can now be defined as:

$$M' = \{\, m \in M \mid \mathsf{Pop}(f'_m) \neq \emptyset \,\}$$

This is indeed a matching, because exclusion and total role constraints cause the matching conditions of M' to be satisfied.

□

We conclude that the verification of the schema property of global atomic populatability is an NP-hard problem. But the verification can also be done in nondeterministic polynomial time. This can be recognised in the following way. Let m be the maximum of all frequency constraint upperbounds and all cardinalities of enumeration constraints. Elements are forced into object types by total role constraints, but this forcing is bounded by m elements. The verification is done by nondeterministically choosing a population bounded in this way, and then checking the constraints. From this we conclude that the verification problem is NP-complete.

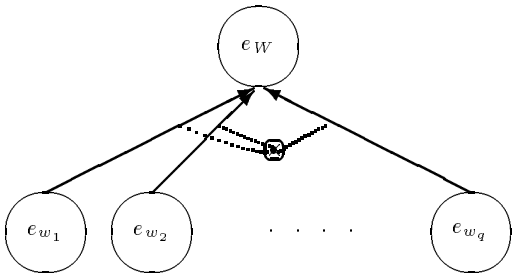


Figure 35: Subtype hierarchy for W

7.3 Global Fact Populatability

Next we consider the verification of the property of global fact populatability, and describe a transformation of a matching problem $\mathcal{M}$ into a schema $\Sigma(\mathcal{M})$. The first step is as in the previous section (see figure 32).

We proceed with the creation of subtype hiearchies for the entity types associated with the elements of W, X and Y. Figure 35 shows the hierarchy for W.

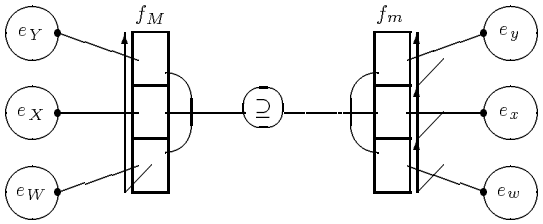


Figure 36: A fact type for each element of M

Next the fact types f_m ($m \in M$) are incorporated as in the previous section. We also introduce a fact type f_M as a placeholder for the elements of M (see figure 36). The fact types f_m and f_M are related via a subset constraint, expressing that each element of f_m should be contained in f_M. However, f_M is restricted to precisely these elements by enforcing an enumeration as shown in figure 37.

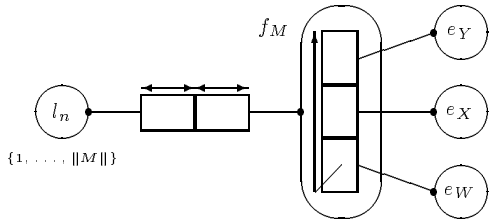


Figure 37: A fact type for the set M

With M' we associate a fact type $f_{M'}$ in the same way as we introduced f_M for M. The special conditions of the matching problem are now represented by a combination of total role and uniqueness constraints (see figure 38). Note that the subset constraint enforces that all instances of $f_{M'}$ occur in f_M. The following theorem is obvious:

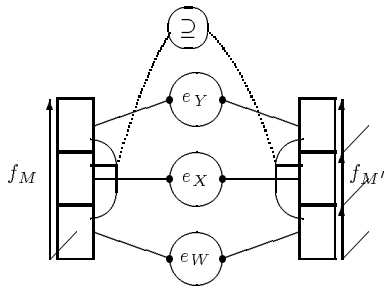


Figure 38: A fact type for the matching

Theorem 7.2

$$\mathcal{M} \textit{ has a matching } \Leftrightarrow \mathsf{GlobFactPop}(\Sigma(\mathcal{M}))$$

Also in this case it is easily checked that verification can be done in nondeterministic polynomial time, which proves the NP-completeness.

8 Conclusions

In this paper the Predicator Model was introduced as a platform for object-role data models. We indicated how techniques as ER and NIAM can be embedded within the Predicator Model.

A primitive language in the style of relational algebra was introduced to reason about the Predicator Model. The Law of the Excluded Miracle indicated the soundness of this algebra. It states that nothing can be derived from relational expressions operating on the empty population. The concept of population was defined as an instantiation of an information structure. Several types of constraints were introduced of which the semantics were defined by expressing under what conditions a population satisfies that type of constraint. These expressions made use of the previously defined algebraic operators. Further research has to be performed on how to incorporate transition oriented constraints in the Predicator Model.

A special type of constraint was the uniqueness constraint. The most difficult form of this constraint involves objectification. The Uniquest Algorithm was introduced to deal with complex uniqueness constraints. The intermediate steps of this algorithm can be visualised using the Object Relation Network representation. We believe that this algorithm is useful as a teaching aid for explaining the semantics of complex uniqueness constraints. Other types of constraints, such as the total role constraint and the set constraints, were given a more powerful semantics than is usually the case.

Two kinds of identification were distinguished: weak and structural identification. Weak identification is a minimal requirement for identification requiring that in a population different objects can be distinguished on the basis of their properties. Structural identification ensures that every population is weakly identified. This is a property that can be determined from the structural description itself.

We also considered cyclic object structures, that are not allowed in modelling techniques. We showed that there exist cyclic object structures that can be populated (instantiated). We have a strong feeling that these structures are worth to investigate, as they may be useful in the gap between information systems and knowledge based systems. Special attention has to be paid to the identification of objects in cyclic structures.

The constraints turned out to be not powerful enough to exclude the empty population as valid. Non-empty populations could be classified into different categories. On the basis of this classification also a classification of schemas was given. The category of a schema is determined by the categories of populations that are possible. Each schema category requires its own verification

procedure. It was shown that two of the most important verification procedures are NP-complete. This implies that CASE-tool manufacturers should concentrate on finding good heuristics and incremental algorithms, to check these properties.

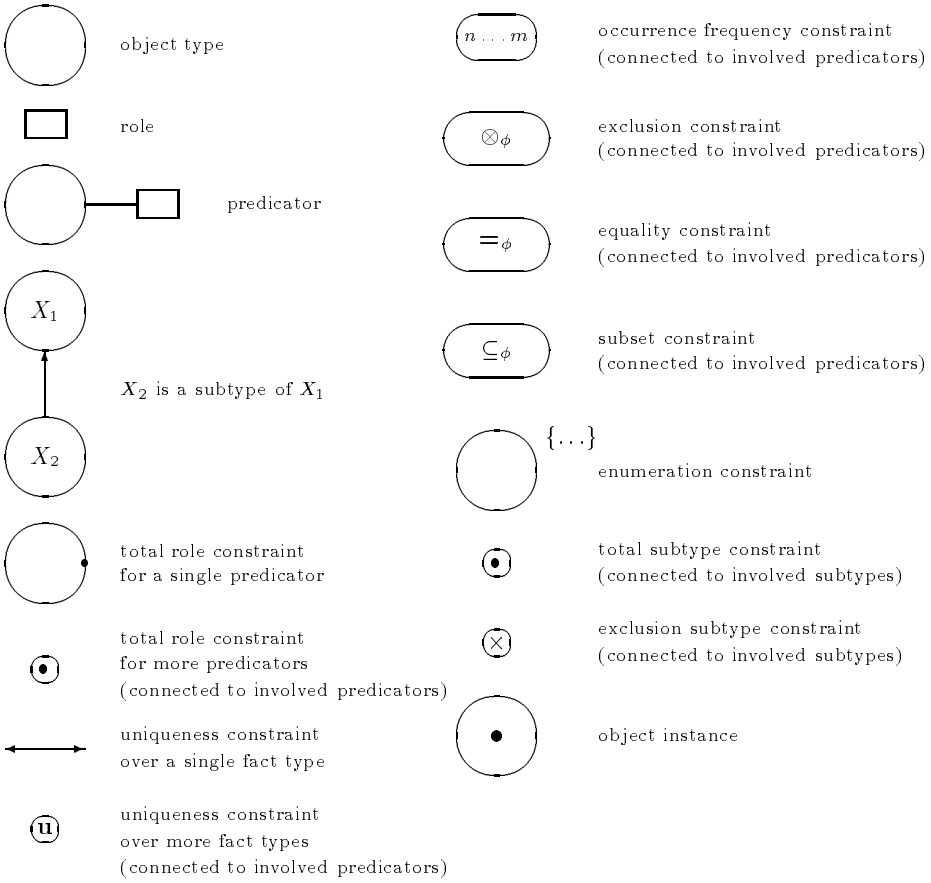
Acknowledgement

We would like to thank Gert Jan Akkerman, Ernst Lippe, Wim Menting, Norbert van Oosterom and Gerard Wijers for their contributive remarks. The remarks of the anonymous referees resulted in many improvements.

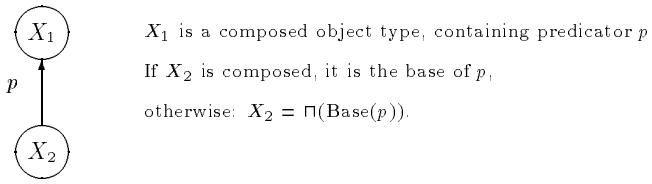
Appendix: Legend of graphical symbols

This appendix contains an overview of the symbols used in this paper.

Graphical symbols for information structure diagrams:



Graphical symbols for Object Relation Network:



References

[1] P.D. Bruza and Th. P. van der Weide. The semantics of data flow diagrams. In *Proceedings of the International Conference on Management of Data*, pages 66–78, 1989. Hyderabad, India.

[2] S. Ceri and G. Gottlob. Translating SQL into relational algebra: optimization, semantics and equivalence of SQL queries. *IEEE Transactions on software engineering*, 11(4):324–345, April 1985.

[3] P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[4] Comtecno. *ProNIAM*Tools*. Comtecno B.V., Vaartweg 86, 1401 RD Bussum, The Netherlands, 1988.

[5] P.N. Creasy. A formalisation of the NIAM conceptual Information Model. Dept of Computer Science, Australian National University, 1983.

[6] Shipman D. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, 1981.

[7] E. D. Falkenberg and Th. P. van der Weide. Formal description of the TOP model. Technical Report 88-01, Vakgroep Informatiesystemen, University of Nijmegen, The Netherlands, 1988.

[8] A.L. Furtado and E.J. Neuhold. *Formal Techniques for Data Base Design*. Springer Verlag, 1985.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. W.H. Freeman and Company, San Fransisco, 1979.

[10] A.H.M. ter Hofstede. Conceptual Task Modelling. Master's thesis, University of Nijmegen, May 1989.

[11] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: Chopping down the methodology jungle. Technical report, Department of Information Systems, University of Nijmegen, The Netherlands, dec 1990. To be published.

[12] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1975.

[13] D. Maier. *The theory of relational databases*. Computer Science Press, 1988.

[14] S.H. Nienhuys-Cheng. Classification and syntax of constraints in binary semantical networks. *Information Systems*, 15(5):497–513, 1990.

[15] G.M. Nijssen and T.A. Halpin. *Conceptual schema and Relational Database Design: A fact oriented approach*. Prentice Hall of Australia Pty Ltd, 1989.

[16] Pandata. *The System Development Workbench*. Pandata B.V., Postbus 1923, 2280 DX Rijswijk, The Netherlands, 1988.

[17] G. Richter and R. Durchholz. IML-Inscribed High-Level Petri Nets. In T.W. Olle, H.G. Sol, and A.A. Verrijn Stuart, editors, *Information System Design Methodologies: A Comparative Review*, pages 335–368. North-Holland, 1982.

[18] H.J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[19] C. Sernadas, J. Fiadeiro, R. Meersman, and A. Sernadas. Proof-theoretic conceptual modelling: The NIAM case study. In *Proceedings of the IFIP TC 8 / WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis*, pages 1–30, 1989.

[20] P. Shoval and M. Even-Chaime. ADDS: A system for automatic database schema design based on the binary-relationship model. *Data and Knowledge Engineering*, 3(2):123–144, 1987.

[21] T.J. Teory, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *Computing Surveys*, 18(2):197–222, 1986.

[22] O.M.F. de Troyer. On rule-based generation of conceptual database updates. In *Proceedings of the IFIP TC 2 Working Conference on Knowledge and Data*, 1986.

[23] O.M.F. de Troyer. RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints. Technical report, Institute for Language Technology and Artificial Intelligence ITK, Tilburg University, 1989. ITK Research Report No. 3.

[24] O.M.F. de Troyer, R. Meersman, and F. Ponsaert. RIDL user guide, febr 1984. Research Report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels.

[25] G.M.A. Verheijen and J. van Bekkum. NIAM: an Information Analysis Method. In T.W. Olle, H.G. Sol, and A.A. Verrijn Stuart, editors, *Information System Design Methodologies: A Comparative Review*, pages 537–590. North-Holland, 1982.

[26] Y. Wand and R. Weber. An ontological analysis of some fundamental information systems concepts. In *Proceedings of the Ninth International Conference on Information Systems*, pages 213–226, Nov 1988.

[27] Th. P. van der Weide. Formal semantics for extended RIDL. Technical Report 87-11, Vakgroep Informatiesystemen, University of Nijmegen, The Netherlands, 1987.

[28] Th. P. van der Weide. Recursive NIAM schemas considered charmful. Technical Report 87-12, Vakgroep Informatiesystemen, University of Nijmegen, The Netherlands, 1987.

[29] Th.P. van der Weide, A.H.M. ter Hofstede, and P. van Bommel. The uniquest algorithm: A formal semantics of complex uniqueness constraints. Technical Report 90-19, Department of Information Systems, University of Nijmegen, The Netherlands, October 1990. To be published.

[30] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice.* Kluwer Academic Publishers, 1990.

Contents