A Hybrid Representation of Vague Collections for Distributed Object Management Systems

Oliver Haase and Andreas Henrich

Abstract—An important characteristic of distributed object management systems is that due to network or machine failure, the environment may become partitioned into subenvironments that cannot communicate with each other. In some application scenarios, it is important that the subenvironments remain operable even in this case. In particular, queries should be processed in an appropriate way. To this end, the final and all intermediate results of a query in a distributed object management system must be regarded as potentially vague. In this paper, we propose a hybrid representation for vague sets and vague multisets designed for this application context. The representation consists of an enumerating part, which contains the elements we could access during query processing, and a descriptive part, which describes the relevant elements we could not access. We introduce Propagation Rules which can be used to minimize the vagueness of a query result represented in this hybrid way. The main advantage of our approach is that the descriptive part of the representation can be used to improve the enumerating part during query processing.

Index Terms—Distributed systems (H.2.4), distributed databases (C.2.4), query processing (H.2.4), query languages (H.2.3), vagueness, inaccessibility.

1 INTRODUCTION—MOTIVATION

In recent years, distributed object management systems (OMS) have received increasing attention, because the distribution reflects the real working situation in various fields of application. Usually the distribution of the system is hidden from the user. If the user issues a query to the system, he normally receives an answer transparently based on the whole distributed object base as if it were completely local. To achieve such a behavior usually the accessibility of all parts of the object base is demanded when processing a query. However, an aspect of distributed OMS which must not be neglected is that the environment may become partitioned into subenvironments that cannot communicate with each other—for example, due to network or machine failure. The usual reaction when part of the object base is inaccessible during query processing is to wait until the respective part becomes accessible again. However, there are situations where such a strategy is inappropriate:

- In some distributed OMS inaccessibility of part of the object base is considered a usual situation. One example in this respect is the OMS of PCTE [18], [9], [28], the ISO and ECMA standard for a public tool interface for software development environments. A further example arises in mobile environments, as described in [16]. The PCTE standard [18] describes the distribution model as follows: “PCTE is based on a community of workstations of possibly differing types connected together by a network. The community is normally seen by the user as a single environment […], though in some circumstances a PCTE installation may be temporarily divided into separated partitions, each of which supports useful work.” The basic idea behind this distribution model is that the data relevant for the current work of a person usually comprises only a small part of the object base which is (mostly) available locally. In this scenario it would, for example, be conceivable to remove part of the object base located on a portable workstation from the network temporarily, to continue work disconnected in a meeting with experts from the application area.

- Whereas in the above example scenario the motivation for queries on partly inaccessible object bases has been the high probability of a fragmentation of the environment, in other scenarios the motivation might be the high urgency of the queries. An example would be a decision support system for brokers in an international bank. Faced with a concrete bid, these people have to come to a decision immediately. In such a situation, a decision support system based on a distributed OMS has to respond to a corresponding query directly. Here we cannot wait until a line which is temporarily down will have recovered. The system has to calculate an answer which is as accurate as possible based on the information actually accessible. Furthermore, it has to offer meaningful information about the reasons...
and the degree of the potential incorrectness in the result.

- A third application scenario where queries on partly accessible object bases would be useful is the worldwide web. In recent years, various query systems for the worldwide web have been proposed. For example, in [20] an SQL-oriented query language for the worldwide web is introduced, but the aspect that the underlying network is relatively unstable is not considered in this approach. Obviously, a declarative query language for the web should consider the potential inaccessibility of part of the system in an appropriate way.

Faced with this scope of applications, techniques for the evaluation of queries on partly inaccessible object bases represent a vital research topic for distributed OMS: On the one hand, an expressive error report has to be assigned to the computed result to clarify the degree of the possible incorrectness and the reasons to the user. On the other hand, adequate error Propagation Rules have to be used that keep the error as small as possible.

A first approach to expressive error reports has been presented in [15], where information about inaccessible parts of the object base, stating the reasons for failure and naming the inaccessible parts, is attached to each affected component of the result. In the present paper, we deal with the problem of error propagation, with the goal to minimize the effects of errors caused by the inaccessibility of part of the object base. The paper elaborates and extends the first ideas presented in [14].

2 Basic Concepts

As mentioned above, the result of a query—and all intermediate results during query processing—becomes potentially vague when part of the object base is inaccessible. Since the result of a declarative query will usually be a collection, this means that we have to deal with vague collections when processing a query in a potentially fragmented environment. Most query languages provide at least three different types of collections: *sets*, *multisets*, and *lists*. In our case, this means that we have to deal with *vague sets*, *vague multisets*, and *vague lists*. To this end, we need

- convenient representations for *vague sets*, *vague multisets*, and *vague lists*, and
- rules for the processing of vague collections. This comprises corresponding extensions for the base operations on vague collections, such as set union and set intersection. Furthermore, general mechanisms are needed to extend the usual query language operators, such as selection, to vague collections.

In the present paper, we concentrate on the representation and the processing rules for vague sets and vague multisets. Vague lists induce serious supplementary problems, because they require additional information about the positions of their elements, which in turn are vague. Therefore, vague lists are an interesting research topic we are concerned with but do not address in this paper.
Fig. 2. Two vague sets.

enumerated and 2) potentially further elements which are not explicitly known. To deal with this potential incompleteness of the upper bound, it would at least be necessary to mark the explicitly given upper bound as complete or incomplete, respectively. This would result in a representation for vague sets consisting of three parts: 1) a lower bound containing all elements which are surely contained in the set, 2) an explicit part of the upper bound, which contains all explicitly known elements which might be contained in the set, and 3) a flag stating whether the explicit part of the upper bound is complete.

The flag in the third part of this representation can be beneficial for the final result of a query as well as for intermediate results:

1. If a vague set \( V \) represents the final result of a query, the user should be aware that there are potentially some elements missing in the stated upper bound.
2. If a vague set \( V \) represents an intermediate result, knowledge about the completeness of the upper bound is important in order to minimize the error in the final result. Let us consider two small examples to demonstrate this point: First, assume a query calculating the intersection of two vague sets \( V \) and \( W \). Let \{Ferrari, Volvo\} be the lower bound of \( V \) and \{Ferrari, Volvo, Porsche\} be the explicit part of the upper bound of \( V \). Furthermore, let \{Ferrari\} be the lower bound of \( W \) and \{Ferrari, Ford, Boeing\} be the explicit part of the upper bound of \( W \). This situation is illustrated in Fig. 2.

If the explicit part of the upper bound of \( V \) is complete, only elements contained in this explicit upper bound can be in the result of the intersection \( V \cap W \). As a consequence, neither Ford nor Boeing can be in \( V \cap W \). On the other hand, if the explicit part of the upper bound of \( V \) was incomplete, Ford and Boeing could be in \( V \cap W \).

Another example query might check whether the element Boeing is contained in the vague set \( V \) given above. If the explicit part of the upper bound of \( V \) is complete, Boeing is obviously not in \( V \). If the explicit part of the upper bound of \( V \) is incomplete, Boeing could be in \( V \), and hence, the query must yield the value unknown.

To further improve the representation of vague sets, we do not simply state that the explicit upper bound is complete or incomplete, but we describe the potentially missing elements by the help of a three-valued logical predicate. We call this predicate the descriptive part of the vague set.

The variety of possible descriptive parts is large: The least precise predicate would yield the value unknown for all elements not contained in the explicit upper bound. This predicate simply represents the incomplete part of the vague set. More sophisticated possibilities can exploit the properties of the desired elements defined in the query or additional knowledge such as meta information about the distribution of the data. In the present paper, we will demonstrate how such predicates can be derived.

Roughly speaking, we employ the descriptive part of a vague set to minimize the vagueness in the result of a query during query processing. This can be demonstrated by the help of the above examples: Assume that the descriptive part of the vague set \( V \) yields unknown only for car manufacturers and false for other companies.

For the first query calculating \( V \cap W \), this means that we need not consider Boeing, which is an element of the explicit upper bound of \( W \), for the result of \( V \cap W \), because the descriptive part of \( V \) yields false for Boeing. The situation with Ford is different, because the descriptive part of \( V \) yields unknown for Ford. Hence, Ford could be a member of the exact set represented by \( V \) and can therefore be part of the result of \( V \cap W \).

For the second query—checking if Boeing is contained in \( V \)—the descriptive part of \( V \) allows us to return false, because the descriptive part itself yields false for Boeing.

It has to be mentioned that the choice of the precision of the descriptive part must always be a compromise. On the one hand, a more elaborate descriptive part can improve the accuracy of the result of a query. On the other hand, it can also increase the query processing effort significantly. In the present paper, we will describe how the accuracy of the descriptive part can be trimmed for the individual requirements.

The above considerations sketch the usefulness of our hybrid representation of vague sets. In order to use this representation for intermediate and final results during query processing, we have to define rules to combine vague sets by extensions of the usual set operators like \( \cap \), \( \cup \) and \( \setminus \). Furthermore, we need rules to handle the typical query language operators such as selection. To this end, we will show in Section 5 how to propagate the vagueness expressed by vague sets through the different operations.

Overall, the paper is organized as follows: First, we describe the concrete environment of our considerations in some detail, to introduce the data model and the distribution model underlying our approach, and to further clarify the motivation for our work. Thereafter, we formally introduce our hybrid representation for vague sets. We propose the Propagation Rules for vague set operations and present the techniques to create and to revise the descriptive part of a vague set in detail. Then, we sketch the situation for vague multisets. We propose their representation and the operations without giving the correctness proofs, because they are analogous to the ones for vague sets. Hereafter, we describe how our hybrid representation can serve as the basis for the implementation of a query
language for a distributed database. Finally, a survey of related work is given and the paper is concluded by a summary and a short outlook concerning further research topics.

3 EXAMPLE ENVIRONMENT

In this section, we introduce the concrete environment which has been the starting point for our considerations. This environment will be used for the examples throughout the paper. It consists of PCTE, the ISO, and ECMA standard for an open repository [18], [9], [28], and of NTT [13], an algebraic set-oriented query language for PCTE. Though we base our considerations on this environment, it should become obvious that they neither depend on a special query language nor on a given data model.

The OMS of PCTE has a semantic data model, based on the entity relationship model. An object base consists of objects, which are connected by links. Objects are typed. The object type hierarchy builds a directed acyclic graph with one root, the object type object. In the usual way, an object of a type ot automatically is also of all types that lie on a path between object and ot. Object types determine the set of applied attributes which are atomic, and the set of applicable link types.

A link type specifies, among other things, a list of attribute types called key attributes, a set of nonkey attribute types, a set of allowed destination object types and its reverse link type; that is because links normally form bidirectional relationships by having a reverse link.

Since the PCTE-OMS is designed as a distributed OMS, objects and links are stored on segments and there is no guarantee that all segments are accessible at a given time. The distribution of the objects over the segments is controlled by the user of the OMS. In PCTE, a new object is always created together with a so-called existence link pointing to the new object. If nothing else is specified, the new object is stored on the same segment as the origin object of this existence link. On the other hand, PCTE offers special operations to create new segments, and the user can explicitly define the segment on which the new object has to be stored. The background behind this distribution technique is that the “user” of the PCTE-OMS will normally be a software development tool. It is the responsibility of this tool to distribute the objects over segments in a way that ensures a high locality when working with the software development environment.

In Fig. 3, a small part of a PCTE schema is given.

The schema consists of four object types: polit_unit, city, capital and country with the attributes name, inhabitants, time_zone, and extent. Subtype relationships between object types are shown by broad shaded arrows. Two pairs of link types are given: The one pair consists of the link types has_capital from country to capital and capital_of from capital to country, and the other pair consists of the link type lies_in from city to country and has_city from country to city. A double arrowhead at the end of a link indicates that the link type has cardinality many. Link types with cardinality many must have a key attribute. In the example, the numeric attribute no is used for this purpose. The link type has_city has such a key attribute, and is hence described as “no.has_city.” An instance of this link type can be addressed by its link name which consists of the concrete value for the key attribute and the type name separated by a dot—e.g., “3.has_city.” The same key attribute type is used for the link type borders_to from country to country. For the considerations in this paper, it is convenient to assume that the link type borders_to is itself its reverse link type.1 Links of the types has_capital, capital_of, and lies_in have no key attribute due to their cardinality one. Hence, a link of type lies_in is addressed by the link name “.lies_in.”

A simple object base for this schema might look like the one sketched in Fig. 4. Here countries are given in bold letters and capitals in italics. The solid arrows denote pairs of has_capital / capital_of links. Dashed arrows represent links of type borders_to; we have omitted the values of the key attributes, because they are not used in the following examples. As with the example in Fig. 1, we assume that the sparsely shaded segment is inaccessible at the moment.

The nonnavigating tool tongue (NTT) is an algebraic approach for a set-oriented query language for PCTE. The most basic operator of NTT is the operator ext that computes the extension of an object type. This extension consists of all objects in the object base which are of the given type. For example, the NTT query ext(capital); would compute all capitals in the whole object base.

Another operator considered in the remainder of this paper is the select operator. It returns all objects of the input set which fulfill a given predicate. Predicates can be logically combined, they can be quantified, and they can contain subqueries. The query

1. Actually, PCTE prevents that a link type is itself its reverse link type. Therefore, in our example, an auxiliary link type would be necessary. However, this technical detail would unnecessarily inflate our example.

---

Fig. 3. Example schema.

Fig. 4. Example object base.
select(ext(capital).time_zone = -6);
would compute all capitals lying in time zone “−6.”

A further operator is relatives. This operator allows for the set-oriented navigation from an input object set via a given regular link name to a result object set. The result contains all objects that can be reached from at least one input object via at least one link matching the regular link name. A regular link name is a PCTE link name which may contain the wild cards “*” and “?” to define the allowed key attribute values. Therefore, the query

relatives(select(ext(country),
name = “Germany”),
*.has_city);

would return all German cities.

Finally, the following query T computes all countries with more than 50 million inhabitants that have at least one neighbor:

select(
    relatives(
        relatives(
            select(ext(capital),
                capital.of,
                *.borders.to),
                inhabitants > 50000000);

The query works as follows: Subquery Q collects all objects of type capital. Subquery R contains the objects (countries) reached by a link of type capital.of from an object in the result of subquery Q. Subquery S in turn comprises all objects (countries) reached by a link of type borders.to from an object of the result of R, and T finally computes all objects in the result of S that have more than 50 million inhabitants.

4 Representation of Vague Sets

As mentioned, the present paper deals with the processing of queries in potentially fragmented distributed data-bases. More precisely, the situation is as follows: We are looking for a concrete set A defined by a declarative query on a distributed object base. Because part of the object base is inaccessible while calculating this set A, the result we gain can only be an approximation of A. Such an approximation of a set A is called a vague set in this paper. A vague set can be envisaged as a set containing all sets which may be a correct choice for A. Unfortunately, because part of the object base is inaccessible while calculating this set, we are not in the position to state all elements of V_l and V_u explicitly. Hence, we need a hybrid representation which consists of

1. two sets enumerating the parts of V_l and V_u we explicitly know, and
2. a descriptive function which decides for each element of T whether it is in V_l, in V_u, or outside V_u.

We will denote the enumerated parts of V_l and V_u by V_l and V_u, and the descriptive function δ_V states the relation to the vague set V for all i ∈ T. Let “t” denote the truth value true, “f” denote false, and “u” denote unknown. Then we get δ_V : T → {t, f, u} where

- δ_V(i) ≡ t means: i is surely contained in the set A approximated by V_l, i.e., i ∈ V_l.
- δ_V(i) ≡ f means: i is surely not contained in the set A approximated by V_u, i.e., i ∈ V_u.
- δ_V(i) ≡ u means: i may or may not be in the set A approximated by V, i.e., i ∈ V_u but i ∉ V_l.

For the formal definition of our representation for vague sets, we write V ∈ Set(T), if V is a vague set over T.

Definition 1. The triple V = (V_l, V_u, δ_V) with V_l ⊆ T, V_u ⊆ T, δ_V : T → {t, f, u} is a vague set over the foundation set T (V ∈ Set(T)), iff

V_l ⊆ V_u (1)

∧ ∀i ∈ V_l : δ_V(i) ≡ t (2)
∧ ∀i ∈ V_u \ V_l : δ_V(i) ≡ u. (3)

Furthermore, V_l and V_u are defined as follows:

V_l = {i ∈ T | δ_V(i) ≡ t} (4)
V_u = {i ∈ T | δ_V(i) ≡ f}. (5)

Each A ∈ Set(T) that lies between V_l and V_u is contained in V:

A ∈ V def= V_l ⊆ A ⊆ V_u. (6)

We demonstrate the application of the definition above by a simple example:

Example 1. Assume the NTT query T (explained in Section 3):

2. It has to be mentioned that this definition does not allow for conditions like “either i or j are contained in A.” Such conditions are not covered by our approach for two reasons: On the one hand, conditions of this type seem to be at least extremely unlikely in our application scenario. On the other hand, the consideration of such conditions would inflate our calculus without adding significant expressiveness.

3. Whenever we refer to the logical values of an expression in this paper, we will use the “≡” sign to state the logical equality of two expressions.
we will use the term **example is the simplest possible choice, stating nothing but**

**Besides, we define the predicate $\delta_T(i)$ by**

$$\delta_T(i) = \begin{cases} t, & \text{if } i \in \hat{T}_i \\ u, & \text{otherwise.} \end{cases}$$

This result is determined as follows: The innermost subquery $Q$ only has access to the capitals Rome, Paris, Washington, and Tokyo. Starting from these capitals, subquery $R$ reaches France, USA, and Japan via links of type `capital_of`. Subquery $S$ determines Germany and Canada as bordering countries, because 1) Spain (bordering to France) lies on the inaccessible part of the object base, and 2) Japan has no bordering country. In the last step, Canada is removed, because it has too few inhabitants.

Since the innermost subquery $Q$ already has produced an incomplete result because of the inaccessibility of part of the object base, we know that the result of $T$ is also incomplete. Due to the trivial descriptive function $\delta_T$, each exact set $A$ that contains the element Germany is contained in $T$. In the following sections, we will show how more expressive descriptive functions can be derived.

The trivial descriptive function $\delta_T$ used in the above example is the simplest possible choice, stating nothing but the fact that the vague set $T$ is incomplete. In the following, we will use the term $\mu_V$ for this descriptive function

$$\mu_V(i) = \begin{cases} t, & \text{if } i \in \hat{V}_i \\ u, & \text{otherwise.} \end{cases}$$

Besides, we define the predicate $\kappa_V$ by

$$\kappa_V(i) = \begin{cases} t, & \text{if } i \in \hat{V}_i \\ f, & \text{if } i \notin \hat{V}_u \\ u, & \text{otherwise.} \end{cases}$$

The descriptive function $\kappa_V$ means that there are no missing elements, i.e., $V$ is complete.

Now that we have defined the hybrid representation for vague sets, we have to adapt the set operations common to query languages to vague sets. These are the usual base operations, like union and intersection, relations, like equality and the subset relation, and query language specific operators, like selection.

### 5 Propagation Rules

Each query language implicitly or explicitly provides a kind of operator determining an initial collection of elements as the starting point for further operators. In general, these operators calculate the extension over a certain type of elements. These may be the tuples of a relation or the objects of a given class or type. To adapt a particular query language to potentially fragmented environments, the semantics of the respective operator must be defined three-valued. In NTT, the described functionality is provided by the operator `ext` that computes the extension of an object type.

Let $\text{type}(i, ot)$ be a three-valued predicate which returns "$t" when the type of $i$ can be determined and is $ot$, "$f" when the type of $i$ can be determined and is not $ot$, and "$u" if the type cannot be determined. Then the vague set $X = \text{ext}(ot)$ can be defined as follows:

$$X_t = \{ i \mid i \text{ is accessible} \land \text{type}(i, ot) = t \}$$

$$X_f = \{ i \mid i \text{ is accessible} \land \text{type}(i, ot) \neq f \}$$

$$\delta_X(i) = \text{type}(i, ot).$$

The following Propagation Rules show how we can combine one or two vague input sets—which can, for example, stem from the application of an extension operator—to a result vague set which is as precise as possible. These Propagation Rules implicitly build a calculus for the stepwise generation of a descriptive function.

#### 5.1 Base Operations on Vague Sets

In this section, we adapt the usual set operations "$\cup", "\cap", and "\" to vague sets. The situation is as follows: Given two vague sets, each of them representing a set of possibly correct sets, we are looking for the vague set that represents the set of all possible result sets with respect to the applied set operation.

When we state the Propagation Rules for the derivation of the result of a three-valued set operator$^5$ $\otimes \in \{ \cup_3, \cap_3, \\setminus_3 \}$, we have to prove the following conditions for the resulting vague set $X = V \otimes W$: First, $X$ has to be a valid vague set, that is, $X$ has to fulfill the Conditions 1, 2, and 3 from the definition of vague sets. These conditions assure the consistency between the explicit parts and the descriptive part of the vague set.

Second, we have to assure the completeness and the strictness of the Propagation Rules. A vague set $X$ is stricter than another vague set $Y$ iff

1. $X_u \setminus X_t \subseteq Y_u \setminus Y_t$ (i.e., if the number of uncertain elements in $X$ is less than in $Y$) and
2. $X_t \supseteq Y_t$ and
3. $X_u \subseteq Y_u$.

$^4$ We use italic capital letters to denote the result of the query with the corresponding bold faced letter.

$^5$ In this paper, we will use the index "3" when we refer to operations on vague sets or to three-valued logical junctions, respectively. If an operator or logical junction is written without index, we always refer to the exact or two-valued case, respectively.
Hence, a vague set \( X = V \cap_3 W \) is strict and complete, if all and only those sets \( C \) are contained in \( X \) for which there is a set \( A \in V \) and a set \( B \in W \) with \( C = A \circ B \). Formally, we can state this as follows:

\[
\forall i : i \in X_U \iff \forall A \in V, B \in W : i \in A \circ B
\]

We can restrict this rule to \( X_U = \{ i \in V_u \cup W_u | \delta_V(i) \equiv t \wedge \delta_W(i) \equiv t \} \). We can restrict this rule to \( X_U = \{ i \in V_u \cup W_u | \delta_V(i) \equiv t \wedge \delta_W(i) \equiv t \} \), because according to Rule 3 in the definition of vague sets, an element \( i \) of \( V_u \cup W_u \) with \( \delta_V(i) \equiv t \) and \( \delta_W(i) \equiv t \) must be at least in the explicit part of the lower bound of the set from which it stems.

Hence, the most extensive explicit parts for \( X = V \cap_3 W \) can be determined by the following expressions:

\[
\hat{X}_U = \{ i \in V_u \cup W_u | \delta_V(i) \equiv f \wedge \delta_W(i) \equiv f \}
\]

In order to state the resulting descriptive function for the vague set intersection we use the three-valued logic given in Table 1.

Now the descriptive part of \( X = V \cap_3 W \) can be expressed as the three-valued conjunction of the descriptive functions of the two vague input sets:

\[
\hat{\delta_X}(i) \equiv \delta_V(i) \wedge_3 \delta_W(i).
\]

The application of these rules is shown in Example 2.

Example 2. Assume that we want to calculate the intersection of two vague sets \( T \) and \( U \). We adopt the vague set \( T \) from Example 1, except for a more precise descriptive function. It requires a candidate for \( T \) to be of type country:

\[\hat{T}_1 = \{ \text{Germany} \} \]

\[\hat{T}_u = \{ \text{USA, Germany, Rome, Japan} \} \]

The computation of \( X = T \cap_3 U \) yields

\[\hat{X}_1 = \{ \text{USA, Germany, Japan} \} \]

To determine \( X_U \) we consider all elements in \( \hat{T}_1 \cup \hat{T}_u \).

Since there is no element with \( \delta_T(i) \equiv t \) and \( \delta_U(i) \equiv t \) we obtain the empty set for \( X_U \).

To determine \( X_u \) we consider all elements in \( \hat{T}_u \cup \hat{T}_1 \).

Since \( \delta_T(\text{Rome}) \equiv f \), \text{Rome} is not included in \( X_u \). For all other elements \( \delta_T \) and \( \delta_U \) yield at least the value “u”, and hence, these elements are contained in \( X_u \).

Note that \( \text{Rome} \) would be contained in \( X_u \) if we used a descriptive function for \( T \) which does not perform the type test. So \( X_u \) is an example for a situation where the descriptive part helps to reduce the vagueness in the result.

At first glance, it may astonish that we yield \( \hat{\delta}_X = \kappa_X \), denoting that the explicit parts of the result are complete.
But since we calculate the intersection of a completely enumerated set \(U\) and another set, the result can obviously only contain elements which are explicitly given in \(\hat{U}\).

Though the above example demonstrates that the descriptive part of our representation can improve the explicit parts, it may not be obvious that propagation rule 9 can yield better results than the simple rule \(\bar{X}_t = \hat{V}_t \cap \hat{W}_t\). However, there are situations where we can obtain \(\delta_t(i) \equiv t\) for an element \(i \notin \hat{V}_t\). And in this case, the propagation rule 9 will include this element into \(X\) whereas the simple rule \(\bar{X}_t = \hat{V}_t \cap \hat{W}_t\) will not.

**Example 3.** Let us consider the following NTT query \(R\) determining all countries which have a capital:

\[
\text{relatives(ext(capital), .capital.of)};
\]

If we evaluate this query on our example object base in Fig. 4, we would achieve the following vague result set \(R\):

\[
\hat{R}_t = \{\text{USA, Japan, France}\}
\]

\[
\hat{R}_u = \{\text{USA, Japan, France}\}
\]

\(\delta_t(i) \equiv \text{type}(i, \text{country})\)

\(\wedge \delta_t(i) \text{ has an outgoing link of type has\_capital.}\)

The explicit parts of \(R\) do not contain Canada and Portugal, because the capitals of these countries are located on the inaccessible segment. The descriptive function \(\delta_t\) exploits the schema information which says that if an object of type country has an outgoing link of type has\_capital, there must be an incoming link of type capital\_of originating from an object of type capital.

Now assume that we want to build the intersection of \(R\) and another vague set \(Z\) with Canada \(\in Z\). In this case, we get \(\delta_t(\text{Canada}) \equiv t\) and \(\delta_t(\text{Canada}) \equiv t\).

Hence, we can include Canada in the explicit lower bound of the result, although Canada has not been in the explicit lower bound of one of its operands.\(^6\)

In order to prove the validity of the rules for the vague set intersection we have to show that the result is in fact a vague set according to Definition 1. To this end, we show Conditions 1, 2, and 3.

**Proof 1 (validity).**

1. **Condition 1:**

   \[
   i \in \bar{X}_t
   \]

   \[
   \Leftrightarrow \quad \delta_t(i) \equiv t \wedge \delta_t(i) \equiv t
   \]

   \[
   \Leftrightarrow \quad (\forall A \in V : i \in A) \wedge (\forall B \in W : i \in B)
   \]

   \[
   \Leftrightarrow \quad \forall A \in V, B \in W : i \in A \cap B.
   \]

2. **Condition 2:**

   \[
   i \in \bar{X}_t
   \]

   \[
   \Rightarrow \quad \delta_t(i) \equiv t \wedge \delta_t(i) \equiv t
   \]

   \[
   \Leftrightarrow \quad (\forall A \in V : i \in A) \wedge (\forall B \in W : i \in B)
   \]

   \[
   \Leftrightarrow \quad \forall A \in V, B \in W : i \in A \cap B.
   \]

3. **Condition 3:**

   \[
   i \in \bar{X}_u \backslash \bar{X}_t
   \]

   \[
   \Leftrightarrow \quad (\forall A \in V : i \notin A) \wedge (\forall B \in W : i \notin B)
   \]

   \[
   \Leftrightarrow \quad \forall A \in V, B \in W : i \notin A \cap B.
   \]

The completeness and the strictness of our rules can be shown by proving Conditions 7 and 8.

**Proof 2 (completeness and strictness).**

1. **Condition 7:**

   \[
   i \in X_t
   \]

   \[
   \Leftrightarrow \quad (\forall A \in V : i \in A) \wedge (\forall B \in W : i \in B)
   \]

   \[
   \Leftrightarrow \quad \forall A \in V, B \in W : i \in A \cap B.
   \]

2. **Condition 8:**

   \[
   i \notin X_u
   \]

   \[
   \Leftrightarrow \quad (\forall A \in V : i \notin A) \wedge (\forall B \in W : i \notin B)
   \]

   \[
   \Leftrightarrow \quad \forall A \in V, B \in W : i \notin A \cap B.
   \]

\(^6\) Of course, this example exploits some peculiarities of the PCTE data model. However, similar situations can arise, for example, also in the ODMG data model [3], where reverse links are also available.
Finally, we have to show that the explicit parts of the vague result set created by our Propagation Rules are as extensive as possible. For the upper bound this is obvious because the rule is based on all elements in $\hat{V}_u \cup \hat{W}_u$. However, if the upper bound is as extensive as possible, Condition 3 from the definition of vague sets, which has been proven for our Propagation Rules above, implies that the lower bound is as extensive as possible, too.

5.1.2 Vague Set Union ($X = V \cup_3 W$)

For the union of two vague sets the result set can be determined by the following rules:

$$X_t = \{ i \in \hat{V}_t \cup \hat{W}_t \mid \delta_V(i) \equiv t \lor \delta_W(i) \equiv t \}$$  \hfill (12)

$$X_u = \hat{V}_u \cup \hat{W}_u$$ \hfill (13)

$$\delta_X(i) \equiv \delta_V(i) \lor \delta_W(i).$$ \hfill (14)

Note that the straightforward rule for $X_u$ would be $X_u = \{ i \in \hat{V}_u \cup \hat{W}_u \mid \delta_V(i) \lor \delta_W(i) \equiv f \}$. Since Conditions 2 and 3 assure that for any $i \in \hat{V}_u \cup \hat{W}_u$ either $\delta_V(i) \equiv f$ or $\delta_W(i) \equiv f$, this can be simplified to $X_u = \hat{V}_u \cup \hat{W}_u$.

The proofs for validity, completeness, and strictness of Rules 12, 13, and 14 are omitted here, because they are analogous to those for the vague set intersection.

5.1.3 Vague Set Difference ($X = V \setminus_3 W$)

The following rules can be used to determine the result of $X = V \setminus_3 W$:

$$X_t = \{ i \in \hat{V}_t \mid \delta_V(i) \equiv t \}$$  \hfill (15)

$$X_u = \{ i \in \hat{V}_u \cup \hat{W}_u \mid \delta_V(i) \equiv f \land \delta_W(i) \equiv f \}$$ \hfill (16)

$$\delta_X(i) \equiv \delta_V(i) \land \neg \delta_W(i).$$ \hfill (17)

The basis for the rule for the explicit part of the lower bound of $X$ is the expression

$$X_t = \{ i \in \hat{V}_t \cup \hat{W}_t \mid \delta_V(i) \equiv t \land \delta_W(i) \equiv t \}.$$  \hfill (18)

However, we need not consider the elements in $\hat{V}_u \setminus \hat{V}_t$ because according to Rule 3 for these elements, we get $\delta_V(i) \equiv u$. Furthermore, we need not consider the elements in $\hat{W}_u$ because according to Rules 2 and (3) for these elements, we get $\delta_W(i) \neq f$.

As with the vague set union the proofs for validity, completeness, and strictness of Rules 15, 16, and 17 are completely analogous to those for the vague set intersection, and are hence omitted here.

We demonstrate the application of the above rules in Example 4.

**Example 4.** We apply the rules for the vague set difference to the vague sets $T$ and $U$ of Example 2. The result $X = T \setminus_3 U$ is as follows:

$$X_t = \emptyset$$

$$X_u = \{ \text{Germany, Japan} \}$$

$$\delta_X(i) \equiv \delta_T(i) \land \neg \delta_U(i)$$

$$\begin{cases} t, & \text{if } i \in T_t \\ u, & \text{if } \text{type}(i, \text{country}) \\ f, & \text{otherwise} \end{cases} \land \neg \delta_U(i)$$

$$\equiv \begin{cases} u, & \text{if } \text{type}(i, \text{country}) \land i \notin U_t \\ f, & \text{otherwise}. \end{cases}$$

There are no elements in $X_U$ because $\delta_U$ is not “$f$” for Germany, which is the only element in $T_t$, Rome is not included in $X_U$ because $br(\text{Rome}) \equiv f$, and USA is not included in $X_u$ because $\delta_U(\text{USA}) \equiv t$.

5.2 Relations on Vague Sets

Query languages often use relations on sets to express a selection criterion. To deal with vague sets, we have to adapt these set relations to vague set relations.

Because a vague set is actually representing a set of possible sets, a relation $\circ_3$ on two vague sets has to be interpreted in a three-valued manner:

- If all possible sets in any of the operands fulfill the given relation, we interpret it as “$t$”:
  $$V \circ_3 W \equiv t \Leftrightarrow \forall A \in V, B \in W : A \circ B.$$  \hfill (18)

- If all possible sets in any of the operands do not fulfill the given relation, we interpret it as “$f$”:
  $$V \circ_3 W \equiv f \Leftrightarrow \forall A \in V, B \in W : \neg (A \circ B).$$  \hfill (19)

- And if there are some possible sets that fulfill the relation, while others do not, we interpret it as “$u$”.

To show the validity of a three-valued relation, it is sufficient to show the validity of Conditions 18 and 19.

In order to state the rules for the relations “$=$” and “$\subseteq$.” Table 2 gives the evaluation rules for three-valued quantified predicates.

5.2.1 Equality of Two Vague Sets

Two vague sets $V$ and $W$ are only equal if they are both exact and represent the same set. They are not equal if there is an element $i$ that is contained in every possible set of $V$ (that is, $i \in V_t$), and that is not contained in any possible set of $W$ (that is, $i \notin W_u$), or vice versa. In all other cases, the value of the equality is unknown:

$$(V =_3 W) \equiv (\forall i : \delta_V(i) \equiv_3 \delta_W(i)).$$  \hfill (20)

**Example 5.** The vague sets $T$ and $U$ given in Example 2 are eventually equal, that is, $T =_3 U \equiv u$, because

7. A vague set $V$ is called exact, if there is no vague set $W$ that is stricter than $V$ (i.e., if $V_t = V_u$).
1. The only element $i$ with $\delta_V(i) \equiv t$ is Germany and $\delta_W(Germany) \equiv u$;
2. The only element $i$ with $\delta_W(i) \equiv t$ is USA and $\delta_T(USA) \equiv u$;
3. Due to 1 and 2: $\forall i: (\delta_V(i) \Leftrightarrow \delta_W(i)) \equiv t$;
4. In addition, 1 and 2 imply:

\[ \exists i : (\delta_V(i) \Leftrightarrow \delta_W(i)) \neq t, \]
5. Due to 3 and 4: $(\forall s:i : \delta_V(i) \Leftrightarrow \delta_W(i)) \equiv u$.

In order to prove the correctness of Rule 20, we have to show the validity of Conditions 18 and 19.

Proof 3 (correctness).

1. Condition 18:

\[ (V =_3 W) \equiv t \]

\( \equiv (\forall s:i : \delta_V(i) \Leftrightarrow \delta_W(i)) \equiv t \)

(20)

2. Condition 19:

\[ (V =_3 W) \equiv f \]

\( \equiv (\exists s:i : \delta_V(i) \Leftrightarrow \delta_W(i)) \equiv f \)

(20)

\( \equiv (\forall A \in V, B \in W : A = B). \)

5.2.2 Subset Relation

A vague set $V$ is a subset of another vague set $W$, if all elements $i$ that are contained in at least one possible set of $V$ (that is, $i \in V_i$) are also contained in every possible set of $W$ (that is, $i \in W_j$). $V$ is no subset of $W$, if there is an element $i$, that is contained in every possible set of $V$ (that is $i \in V_i$), and that is contained in no possible set of $W$ (that is, $i \notin W_j$).

5.3 Query Language Operators

The usual set operators, adapted to vague sets above, are part of most query languages; nevertheless, they are no special feature of query languages. In this section, we focus on typical query language operators. To obtain a high degree of generality, we keep these operators as abstract as possible, and illustrate them giving concrete examples in NTT. We treat two kinds of operators: selection and the so-called fold operator.

5.3.1 Selection

Every query language uses selection as a separate operator like the relational algebra or NTT or implicitly, for example, integrated in the where-clause of a query in SQL, the ODMG proposal OQL [3] or P-OQL [15].

In the two-valued case, selection $\sigma$ is a mapping

\[ \sigma : \text{Set}(T) \times (T \rightarrow \{t, f\}) \rightarrow \text{Set}(T) \]

\[ \sigma(A, P) = \{i \in A \mid P(i)\}. \]

When we deal with vagueness and three-valued logic, selection is a mapping

\[ \sigma_3 : \tilde{\text{Set}}(T) \times (T \rightarrow \{t, f, u\}) \rightarrow \tilde{\text{Set}}(T) \]

A vague selection $\sigma_3$ gets both a vague set $V$ and a three-valued predicate $P_3$ as input and returns a vague set $X = \sigma_3(V, P_3)$ as output. This result set can be determined as follows:

\[ \tilde{X}_i = \{i \in V_i \mid P_3(i) \equiv t\} \]

\[ \tilde{X}_u = \{i \in V_u \mid P_3(i) \equiv f\} \]

\[ \delta_X(i) \equiv \delta_V(i) \land P_3(i). \]

The application of the above rules is shown in Example 6.

Example 6. Assume the vague set $V$ with
the case without vagueness, will see in the next section. Besides selection, there is no other operator with a generality like this. But concerning a lot of existing query languages, we think there is one common principle behind several important operators: namely that a set operator actually is defined via its effect on the individual elements of the input set, and the result is built as the union of the results for the single elements.

We exploit this general principle to define the fold operator \( \phi \) containing all these operators as instances of itself: The effect on a single element can be represented as a mapping \( F : T_1 \rightarrow \text{Set}(T_2) \), which is used as an input to the fold operator in order to constitute a concrete operator. In the case without vagueness, \( \phi \) is defined as:

\[
\begin{align*}
\phi &: \text{Set}(T_1) \times (T_1 \rightarrow \text{Set}(T_2)) \rightarrow \text{Set}(T_2) \\
\phi(A,F) &= \{ i \in T_2 \mid \exists j \in A : i \in F(j) \}.
\end{align*}
\]

The sketch in Fig. 5 illustrates the principle of the fold operator without vagueness.

If we take into account that both the input set and the mapping can be vague, we obtain the three-valued fold operator \( \phi_3 \):

\[
\phi_3 : \text{Set}(T_1) \times (T_1 \rightarrow \text{Set}(T_2)) \rightarrow \text{Set}(T_2).
\]

The three-valued fold operator uses a vague set and a vague mapping as input and returns a vague set as output.

Before giving the rules for the vague fold operator, we define what we mean by calling a mapping vague.

**Definition 2.** A mapping \( F_3 : T_1 \rightarrow \text{Set}(T_2) \) is called a vague mapping. It contains all mappings \( F : T_1 \rightarrow \text{Set}(T_2) \) fulfilling the condition: \( \forall i \in T_1 : F(i) \in F_3(i) \).

Now we can state the rules for the vague fold operator \( X = \phi_3(V, F_3) \):

\[
\begin{align*}
\hat{X}_l &= \{ i \mid \exists j \in \hat{V}_l : i \in F_3(j) \} \\
\hat{X}_u &= \{ i \mid \exists j \in \hat{V}_u : i \in F_3(j) \} \\
\delta_X(i) &= \exists j : \delta_3(j) \land \delta_3(i).
\end{align*}
\]

Next we give two examples that show the use of the fold operator.

**Example 7.** First, we show that the selection operator can be envisaged as an instance of the generic fold operator. To this end, we have to define a mapping \( F_3 : T \rightarrow \text{Set}(T) \) which represents the predicate \( P_3 \) of the three-valued selection:

\[
\begin{align*}
F_3(j)_l &= \begin{cases} 
\{ j \}, & \text{if } P_3(j) \equiv t \\
\emptyset, & \text{otherwise}
\end{cases} \\
F_3(j)_u &= \begin{cases} 
\{ j \}, & \text{if } P_3(j) \neq f \\
\emptyset, & \text{otherwise}
\end{cases} \\
\delta_{F_3(j)}(i) &= \begin{cases} 
P_3(i), & \text{if } j = i \\
f, & \text{otherwise}
\end{cases}
\end{align*}
\]

Now we can insert this mapping \( F_3 \) into Rule 25 to determine the explicit part of the lower bound of \( X = \phi_3(V, F_3) \):

\[
\hat{X}_l = \{ i \mid \exists j \in \hat{V}_l : i \in F_3(j) \} = \{ i \in \hat{V}_1 : P_3(i) \equiv t \}.
\]

Analogously, we achieve \( \hat{X}_u = \{ i \in \hat{V}_u : P_3(i) \neq f \} \) for the explicit part of the upper bound.

According to (27), we yield the following expression for \( \delta_X(i) \):

\[
\phi_3 : \text{Set}(T_1) \times (T_1 \rightarrow \text{Set}(T_2)) \rightarrow \text{Set}(T_2).
\]

The descriptive part of \( Y \) shows how the selectivity of the predicate \( P_3 \) enriches the descriptive part of the input set \( V \).

We do not need to prove the validity or the completeness and strictness of the rules given above, because selection can be regarded as an instance of the fold operator, as we will see in the next section.

**5.3.2 Fold Operator**

Besides selection, there is no other operator with a generality like this. But concerning a lot of existing query languages, we think there is one common principle behind several important operators: namely that a set operator actually is defined via its effect on the individual elements of the input set, and the result is built as the union of the results for the single elements.

We exploit this general principle to define the fold operator \( \phi \) containing all these operators as instances of itself: The effect on a single element can be represented as a mapping \( F : T_1 \rightarrow \text{Set}(T_2) \), which is used as an input to the fold operator in order to constitute a concrete operator. In the case without vagueness, \( \phi \) is defined as:

\[
\begin{align*}
\phi &: \text{Set}(T_1) \times (T_1 \rightarrow \text{Set}(T_2)) \rightarrow \text{Set}(T_2) \\
\phi(A,F) &= \{ i \in T_2 \mid \exists j \in A : i \in F(j) \}.
\end{align*}
\]
Example 8. To define the mapping $F_3$ used to represent relatives by means of the fold operator, we introduce two conventions:

1. We use the regular link name $RL$ as a synonym for the set consisting of all link names matching $RL$ itself.

2. The notation "$j \xrightarrow{i} i$" means that a link with name $l$ originating from $j$ refers to $i$. In case of inaccessibility, both the link $l$ and the destination $i$ can be inaccessible when we want to navigate from $j$ via $l$ to $i$. Thus, we use the three-valued expression "$j \xrightarrow{3} i$" which yields "$u$" when inaccessibility prevents an exact evaluation.

We obtain the three-valued mapping $F_3$ as follows:

$$F_3(j)_l = \{ i | \exists l \in RL : (j \xrightarrow{l} i) \equiv t \}$$

$$F_3(j)_u = \{ i | \exists l \in RL : (j \xrightarrow{l} i) \not\equiv f \}$$

$$\delta_{F_j}(i) \equiv \exists d \in RL : j \xrightarrow{3} d.$$

In our concrete environment, we can simplify the rule for $F_3(j)_u$ to $F_3(j)_u = F_3(j)_l$, because the determination of possible destination objects without direct navigation over the respective link would be too expensive.

Inserting the above definition of $F_3$ into Rule 25, we get the explicit part of the lower bound of $X = \phi_3(V, F_3) = \text{relatives}_3(V, RL)$:

$$\hat{X}_l = \{ i | \exists j \in \hat{V}_l : i \in F_3(j)_l \} = \{ i | \exists j \in \hat{V}_l, l \in RL : (j \xrightarrow{l} i) \equiv t \}.$$

Analogously, $\hat{X}_u$ can be computed to

$$\hat{X}_u = \{ i | \exists j \in \hat{V}_u, l \in RL : (j \xrightarrow{l} i) \not\equiv f \}.$$

As given in (27), $\delta_X$ is

$$\delta_X(i) \equiv \exists j : \delta_V(j) \land \exists d \in RL : j \xrightarrow{3} d.$$

Now we can apply these rules for the vague version of the relatives operator to the expression

$$Y = \text{relatives}_3(V, \text{borders}_0),$$

where $V$ is the vague set defined in Example 6.

Evaluated according to the accessibility situation depicted in Fig. 4, we obtain

$$\hat{Y}_l = \{ \text{USA, Canada} \}$$

$$\hat{Y}_u = \{ \text{USA, Canada, Germany} \}$$

$$\delta_Y(i) \equiv \exists j : \delta_V(j) \land \exists l \in \text{borders}_0 : j \xrightarrow{l} i$$

$$\equiv \begin{cases} t, & \text{if } \exists j \in \hat{V}_l, l \in \text{borders}_0 : (j \xrightarrow{l} i) \equiv t \\ f, & \text{if } \forall j : \forall l \in \text{borders}_0 : (j \xrightarrow{l} i) \equiv f \\ u, & \text{otherwise}. \end{cases}$$

The explicit parts of $Y$ are computed as follows: $\text{USA}$ is in $\hat{Y}_l$ because it borders $\text{Canada}$, which is a member of $\hat{V}_l$, and $\text{Canada}$ is in $\hat{Y}_l$ because it borders to $\text{USA}$, which is also a member of $\hat{V}_l$. Since $\text{Japan}$ has no bordering countries, it does not contribute any element to $\hat{Y}_l$. The only accessible neighbor of Portugal or France is $\text{Germany}$. Therefore, $\text{Germany}$ is the only additional element in $\hat{Y}_u$.

Analogously, to the base operations we prove the validity, completeness, and strictness of the rules for the vague fold operator $X = \phi_3(V, F_3)$.

**Proof 4 (validity).**

1. **Condition 1:**

   $$i \in \hat{X}_l \iff (25) \exists j \in \hat{V}_l : i \in F_3(j)_l$$
   $$\iff (1) \exists j \in \hat{V}_u : i \in F_3(j)_u$$
   $$\iff (20) i \in \hat{X}_u.$$

2. **Condition 2:**

   $$i \in \hat{X}_l \iff (25) \exists j \in \hat{V}_l : i \in F_3(j)_l$$
   $$\iff (2) \exists j \in \hat{V}_l : \delta_{F_j}(i) \equiv t$$
   $$\iff (2) \exists j : \delta_V(j) \equiv t \land \delta_{F_j}(i) \equiv t$$
   $$\iff (\text{Table 2}) (\exists j : \delta_V(j) \land \delta_{F_j}(i)) \equiv t$$
   $$\iff (27) \delta_X(i) \equiv t.$$

3. **Condition 3:**

   $$i \in \hat{X}_u \setminus \hat{X}_l \iff (25, 26) \exists j \in \hat{V}_u : i \in F_3(j)_u$$
   $$\land \neg (\exists j \in \hat{V}_l : i \in F_3(j)_l)$$
   $$\iff (2, 3) \exists j : \delta_V(j) \not\equiv f \land \delta_{F_j}(i) \not\equiv f$$
   $$\land \neg (\exists j : \delta_V(j) \equiv t \land \delta_{F_j}(i) \equiv t)$$
   $$\iff (\text{Table 1}) \exists j : (\delta_V(j) \land \delta_{F_j}(i)) \not\equiv f$$
   $$\land \neg (\exists j : (\delta_V(j) \land \delta_{F_j}(i)) \equiv t)$$
   $$\iff (\text{Table 2}, 27) \delta_X(i) \equiv u.$$
In order to prove the strictness and the completeness of the rules for the vague fold operator, we have to show that $X$ describes the strictest vague set containing all sets $C$ for which there is a set $A \in V$ and a mapping $F \in F_3$ with $C = \phi(A, F)$. This can be expressed as follows:

\begin{align*}
\forall i : i \in X_I & \Leftrightarrow \forall A \in V, F \in F_3 : \exists j \in A : i \in F(j) \quad (28) \\
\land \forall i : i \notin X_u & \Leftrightarrow \forall A \in V, F \in F_3 : \forall j \in A : i \notin F(j). \quad (29)
\end{align*}

**Proof 5 (completeness and strictness).**

1. **Condition 28:**

\[ i \in X_I \Leftrightarrow \delta_X(i) \equiv t \quad (1) \]

\[ \exists j : (\delta_V(j) \land \delta_F(j)(i)) \equiv t \quad (27) \]

Table 2: \[ \exists j : (\delta_V(j) \land \delta_F(j)(i)) \equiv t \]

Table 1: \[ \exists j : \delta_V(j) \equiv t \land \delta_F(j)(i) \equiv t \]

2. **Condition 29:**

\[ i \notin X_u \Leftrightarrow \delta_X(i) \equiv f \quad (5) \]

\[ \exists j : (\delta_V(j) \land \delta_F(j)(i)) \equiv f \quad (27) \]

Table 2: \[ \forall j : (\delta_V(j) \land \delta_F(j)(i)) \equiv f \]

Table 1: \[ \forall j : \delta_V(j) \equiv f \lor \delta_F(j)(i) \equiv f \]

6. **Levels of Accuracy for the Descriptive Part**

In the previous section, we have implicitly shown how a descriptive function for a complete query can be derived. The starting point has been a vague set stemming from some type of extension operator. Then the descriptive function has been evolved in parallel to the inside-out evaluation of the query. In this way, the complete semantics of the query is transferred into the descriptive part of the result.

In this section, we will first show how meta-information, such as type information and locality information, influences the quality of the descriptive function. Thereafter, we will present a technique called back tracing to adjust the accuracy of the descriptive function. Applying this approach, an individual compromise between the computation effort and the accuracy of the result can be achieved.

6.1 **Typed vs. Type-Free Query Languages**

The most trivial descriptive function for an incomplete vague set $V$ is $\delta_V \equiv \mu V$. This function does not really describes $V$, but says that all elements may be in $V$. All elements actually means all elements $i \in T$, where $T$ is the foundation set of $V$. The smaller the foundation set $T$, the stricter $V$ is described by the descriptive function. The means to restrict the foundation set is, in the context of query languages, to use typing. There is not only the distinction between typing and no typing, but one can distinguish different granularities of typing: If a query language is type free, we know nothing about the missing elements of a vague set. They can be atomic or complex values, or even objects of the object base.

On a coarse-grained level typing distinguishes between different atomic and complex values and objects of the object base. An incomplete vague set built over natural values, for example, misses natural values and nothing else.

On a fine-grained level query languages use value types as above and additionally distinguish different types for the objects of the object base. In this case, if a vague set of objects is incomplete, we know the missing objects to be of the corresponding object type.

NTT, for example, uses for each operator an inference rule to determine the object type for the result vague set, depending on the types of the vague input sets. Here, we do not state these rules explicitly; instead we apply them to our example query $T$ and explain their meaning:

\[
\text{select}(
\text{relatives(}
\text{relatives(}
\text{ext(capital), } Q \}
\text{R})}
\text{S})
\text{T}
\text{inhabitants > 50000000})
\]

The innermost expression $\text{ext(capital)}$ has the object type $\text{capital}$. According to Fig. 3, navigation via links of type $\text{capital_of}$ reaches objects of type $\text{country}$; therefore, subquery $R$ has the type $\text{country}$. The next navigation via links of type $\text{borders_to}$ reaches objects of type $\text{country}$ as
well; therefore subquery $S$ has type $\text{country}$. Finally, the type remains the same after the selection, which means that the complete query $T$ has the object type $\text{country}$. If it is computed incomplete, we will know that only objects of that type can be missing.

Let’s illustrate these considerations by a short example.

**Example 9.** In Example 1, the evaluation of the query $T$ on the partly accessible object base of Fig. 4 has yielded the vague set $T$ with

\[
\hat{T}_i = \{\text{Germany}\},
\hat{T}_a = \{\text{Germany}\}
\]

because we used no information about the missing elements. Now, assume we want to calculate $Z = T \cap_W W$ with

\[
\hat{W}_i = \{\text{Germany}, \text{Japan}, \text{Canada}, \text{USA}, \text{Rome}\},
\hat{W}_a = \{\text{Germany}, \text{Japan}, \text{Canada}, \text{USA}, \text{Rome}\},
\delta_W = \kappa_W,
\]

i.e., $W$ is an exact set.

If we assume coarse-grained typing, we only know both $T$ and $W$ to contain objects of the object base instead of values. Therefore, we must assume the whole object base to be $T_v$. This means in particular that all elements of $W_v$ have to be inserted into $Z^0_v$—note that we use an upper index $v$ to denote the $v$th revision of something.

Applying Rules 9, 10, and 11 for the vague set intersection yields

\[
\hat{Z}^0_i = \{\text{Germany}\},
\hat{Z}^0_a = \{\text{Germany}, \text{Japan}, \text{Canada}, \text{USA}, \text{Rome}\},
\delta^0_Z = \kappa^0_Z.
\]

If we assume fine-grained typing, we know that $T$ has the object type country. Therefore, we know that $T_v$ can only contain objects of type country, which leads to the new revision $Z^1$ with

\[
\hat{Z}^1_i = \{\text{Germany}\},
\hat{Z}^1_a = \{\text{Germany}, \text{Japan}, \text{Canada}, \text{USA}\},
\delta^1_Z = \kappa^1_Z,
\]

because “Rome” does not have the object type country, and hence cannot be contained in $T_v$ or $Z^1_v$.

And indeed, $Z^1$ calculated exploiting type information is stricter than $Z^0$.

### 6.2 Locality Information

In some cases, available locality information about the missing elements of a vague set can be used to improve the quality of the descriptive part. If we can specify the segments of the object base on which potential candidates must reside, we can derive an improved descriptive function $\delta'_y$ from any other descriptive function $\delta_y$. To this end, we first check an element $i \not\in \hat{Y}_u$ against the locality criterion. If $i$ does not reside on one of the specified segments, $\delta'_y(i)$ can be set to “$f$”:

\[
\delta'_y(i) = \begin{cases} 
\delta_y(i), & \text{if } i \text{ resides on a specified segment} \\
\land i \in \hat{Y}_u \\
f, & \text{otherwise.}
\end{cases}
\]

It has to be mentioned that locality information cannot always be used. If, for example, a relatives operator is applied to a vague set $W$ with an incomplete $W_u$, we cannot use this improvement. The reason is that we cannot follow links originating from unknown objects. These links can potentially refer to objects residing on arbitrary segments. On the other hand, the operator ext, like all operators using no navigation, allows the application of the locality improvement, just as the navigation by relatives, if the origin vague set is complete.

### 6.3 Back Tracing

In the section Propagation Rules, we have stated the rules to compute the descriptive part $\delta$ of the result of a query. The application of these rules allows us to compute the best possible descriptive function $\delta$ for the result.\(^8\)

Unfortunately, the descriptive function $\delta$ computed in this way can become complicated for nontrivial queries, and if we take into account that $\delta$ may be used during query processing to check for a lot of elements their relation to the query, it might be interesting to use a less strict descriptive function $\hat{\delta}$ which can be checked with less effort.

One way to achieve such a less strict—and less expensive to compute—descriptive function is to substitute the descriptive function of at least one operand $V$ of an operator by the trivial descriptive function

\[
\mu_V(i) = \begin{cases} 
t, & \text{if } i \in \hat{V}_i \\
u, & \text{otherwise.}
\end{cases}
\]

As a consequence, if we neglect the knowledge about the missing elements of one or more operands, we can only exploit the semantics of the operator itself and, should the occasion arise, the information about the other operand to build the descriptive part of the result.

For example, if we apply a selection with a predicate $P_3$ to a vague input set $V$ and we use $\mu_V$ for the vague input set, the descriptive part of the result $Z$ of the selection can only exploit the predicate $P_3$, and hence, we get:

\[
\beta_Z(i) = \begin{cases} 
t, & \text{if } i \in \hat{Z}_i \\
\land f, & \text{if } P_3(i) \equiv f \\
u, & \text{otherwise.}
\end{cases}
\]

Now let us switch our point of view and take a look at the query from the top, to check whether an element $i$ is part of the result of a query. Then applying the descriptive function derived by our Propagation Rules implicitly means to trace back the query top-down, operator by operator, and to check whether $i$ could have been in the result. Furthermore, using the trivial descriptive function $\mu_V(i)$

---

8. Best, in this case, means strictest, and by the strictness proofs we have shown that the Propogation Rules yield the strictest possible function.
for a subquery means stopping back tracing at this subquery.

For the base operations on vague sets the use of the trivial descriptive function $\mu$ for one operand specializes the corresponding rules for the computation of the descriptive function of the result. As an example, we consider vague set union.

### 6.3.1 Vague Set Union

When we build $X = V \cup W$ with $\delta_V \equiv \mu_V$, we get the following simplified version of Rule 14:

$$\delta_X(i) \equiv \mu_V(i) \lor \delta_W(i)$$

$$\equiv \begin{cases} t, & \text{if } i \in \bar{V}_t \lor \delta_W(i) \equiv t \\ u, & \text{otherwise.} \end{cases}$$

In the following we show how the use of the trivial descriptive function $\mu$ for the operand of a query language operator influences the corresponding Propagation Rules.

### 6.3.2 Selection

If $X = \sigma_3(V, F_3)$, with $\delta_V \equiv \mu_V$, we obtain the following specialized version of Rule 24:

$$\delta_X(i) \equiv \mu_V(i) \land \delta_3(i)$$

$$\equiv \begin{cases} t, & \text{if } i \in \bar{V}_t \land \delta_3(i) \equiv t \\ f, & \text{if } \delta_3(i) \equiv f \\ u, & \text{otherwise.} \end{cases}$$

This rule has an intuitive interpretation: If we know nothing about the missing elements of the vague input set $V$, then 1) only the explicitly given elements of $V$ for which the selection predicate $P_3$ is true ($X_t$ calculated according to Rule 22) are surely contained in the result, 2) all elements for which the predicate is false are surely not in the result, and 3) for all other elements, their relation to the result is unknown.

We show the application of this specialized rule in Example 10.

**Example 10.** Let us return to the vague intersection $Z = T \cap W$ of Example 9. We assume the vague set $W$ and the explicit part of $T$ as given in that example, and we still use type information.

If we want to trace back the query $T$ exactly one level, we have to consider only the outermost operator of $T = \text{select}(S, \text{inhabitants} > 50,000,000)$ by setting the descriptive function of $S$ to $\delta_S \equiv \mu_S$. Hence, the descriptive function for $T$ exploits only the semantics of this outermost operator.

This means we apply the simplified rule for the vague set selection as stated above and obtain the descriptive function

$$\delta_T(i) \equiv \begin{cases} t, & \text{if } i \in \bar{T}_t \\ f, & \text{if } \text{inhabitants}(i) \leq 50,000,000 \\ u, & \text{otherwise.} \end{cases}$$

Now we can use this descriptive function for the propagation Rules 9, 10, and 11 for the vague set intersection. Here, $\delta_T$ helps us to restrict the explicit upper bound of the result, because—in contrast to Example 9, where the semantics of the selection was not exploited—$\delta_T^1(\text{Canada})$ yields "f" because Canada has less than 50 million inhabitants. Hence, Canada need not be inserted in $Z^2_w$. We obtain the new revision of the vague result set $Z = T \cap W$:

$$\hat{Z}^2_w = \{\text{Germany}, \text{Japan, USA}\},$$

$$\delta_Z^2(i) \equiv \delta^2_Z.$$

### 6.3.3 Fold Operator

If $Y = \phi_3(V, F_3)$, with $\delta_V \equiv \mu_V$, we obtain the following specialized version of Rule 27:

$$\delta_Y(i) \equiv \exists_j : \mu_V(j) \land \delta_{F(i)}(j)$$

$$\equiv \exists_j : \begin{cases} t, & \text{if } j \in \bar{V}_t \land \delta_{F(i)}(j) \equiv t \\ f, & \text{if } \delta_{F(i)}(j) \equiv f \\ u, & \text{otherwise} \end{cases}$$

Using this specialized version of Rule 27, we get the specialized rule for the computation of the descriptive part of the three-valued relatives operator.

**Example 11.** The specialized rule for the descriptive part of

$Y = \text{relatives}_3(V, RL)$ is computed as follows:

$$\delta_Y(i) \equiv \begin{cases} t, & \text{if } \exists j \in \bar{V}_t \land \delta_{RL}(j) \equiv t \\ f, & \text{if } \forall j : \delta_{RL}(j) \equiv f \\ u, & \text{otherwise} \end{cases}$$

Let’s have a further look at our running example.

**Example 12.** Now we trace back the query $T$ two levels exploiting the fact that the subquery $S$ has the form $S = \text{relatives}(R, *.borders_to)$. We set $\delta^0_R = \mu_R$ and get the first revision of $\delta_S$:
For other schemas it might be necessary to state this condition explicitly.

In total, the example above shows two facets of the Propagation Rules based on our hybrid representation of vague sets: 1) The use of the descriptive part allows a better (because it is stricter) computation of the result of a query, and 2) the possibility to stop back tracing at an arbitrary level enables us to adjust the accuracy of the result in relation to the corresponding computation costs.

7 Vague Multisets

Now that we have explained our treatment of vague sets exhaustively, we will give a short impression of how our approach can be transferred to multisets.

Analogously to vague sets, a vague multiset can be seen as a set of multisets where each element is a multiset that could be the correct answer to the computed query. We denote a multiset $A$ over the foundation set $T$ ($A \in \text{Bag}(T)$) as a partial mapping $A : T \mapsto N$, where the domain $\text{dom}(A)$ contains all elements which occur in the multiset $A(i)$ is the number of occurrences of $i$ in $A$.

The representation of a vague multiset $V$ over the foundation set $T$ ($V \in \text{Bag}(T)$) is similar to vague sets: The lower and the upper bound of a vague multiset $V$ in $\text{Bag}(T)$ are expressed by two functions $\lambda_V$ and $\nu_V$ which state for each element $i \in T$ its minimum and its maximum number of potential occurrences in $V$; this corresponds to the lower and the upper bound $V_l$ and $V_u$ of a vague set $V$.

To reflect the fact that we cannot enumerate all relevant elements explicitly, we define $\lambda_V$ and $\nu_V$ as total mappings over $T$ and complement these mappings with a subset $\Delta_V$ of $T$ which describes the elements we actually have under access. Since we do not carry those elements for which we know exactly that they are not in $V$, $\Delta_V$ contains only elements $i$ with $\nu_V(i) > 0$. Hence, the restrictions of $\lambda_V$ and $\nu_V$ to the domain $\Delta_V$ correspond to the explicit parts $V_l$ and $V_u$ of a vague set $V$.

In the following, we extend $N$ to the complete lattice $N^\infty = N \cup \infty$. The symbol $\infty$ denotes the maximum element of the lattice $N^\infty$.

Definition 3. The triple $V = (\lambda_V, \nu_V, \Delta_V)$ with $\lambda_V : T \mapsto N$ and $\nu_V : T \mapsto N^\infty$ is a vague multiset over $T$ iff

$\forall i \in T : \lambda_V(i) \leq \nu_V(i)$
\begin{align*}
\wedge \Delta_V & \subseteq T \\
\wedge \forall i \in \Delta_V : \nu_V(i) > 0.
\end{align*}

Each multiset $A$ that lies between $\lambda_V$ and $\nu_V$ is contained in $V$:

$A \in V \iff \forall i \in \text{dom}(A) : \lambda_V(i) \leq A(i) \leq \nu_V(i)$
\begin{align*}
\wedge \forall i \in T \setminus \text{dom}(A) : \lambda_V(i) = 0.
\end{align*}

7.1 Base Operations on Vague Multisets

For a terse presentation, we only state the rules for the combination of two vague multisets without proving their correctness. In principle, the proofs can be done analogously to the ones concerning vague sets.
7.1.1 Vague Multiset Union (X = V ∪_{M, 3} W)

The result X of the vague multiset union V ∪_{M, 3} W can be computed as follows:

\[ \lambda_X(i) = \lambda_V(i) + \lambda_W(i) \]
\[ v_X(i) = \begin{cases} v_V(i) + v_W(i), & \text{if } v_V(i) \neq \infty \land v_W(i) \neq \infty \\ \infty, & \text{otherwise.} \end{cases} \]
\[ \Delta_X = \Delta_V \cup \Delta_W. \]

Since each element i which is in \( \Delta_V \) or in \( \Delta_W \) has a value \( v_V(i) \) or \( v_W(i) \) greater than 0, and therefore also has a value \( v_X(i) \) greater than 0, \( \Delta_X \) is the union of \( \Delta_V \) and \( \Delta_W \).

7.1.2 Vague Multiset Intersection (X = V \cap_{M, 3} W)

The result X of the vague multiset intersection V \cap_{M, 3} W can be computed as follows:

\[ \lambda_X(i) = \min\{\lambda_V(i), \lambda_W(i)\} \]
\[ v_X(i) = \min\{v_V(i), v_W(i)\} \]
\[ \Delta_X = \{i \in \Delta_V \cup \Delta_W \mid v_X(i) > 0\}. \]

Here \( \Delta_X \) is calculated as the subset of all explicitly known elements (\( \Delta_V \cup \Delta_W \)) which fulfill the condition \( v_X(i) > 0 \) given in the definition of vague multisets.

7.1.3 Vague Multiset Difference (X = V \setminus_{M, 3} W)

The result X of the vague multiset difference V \setminus_{M, 3} W can be computed as follows:

\[ \lambda_X(i) = \max\{0, \lambda_V(i) - v_W(i)\} \]
\[ v_X(i) = \max\{0, v_V(i) - \lambda_W(i)\} \]
\[ \Delta_X = \{i \in \Delta_V \cup \Delta_W \mid v_X(i) > 0\}. \]

7.2 Fold Operator

Before defining the fold operator for vague multisets, we first define the fold operator \( \phi_M \) for multisets. Thereafter, the version for vague multisets \( \phi_{M, 3} \) will be given.

The fold operator for multisets is—as in the case for sets—defined as the union of the results for the single elements of the operand:

\[ \phi_M : \text{Bag}(T_1) \times (T_1 \rightarrow \text{Bag}(T_2)) \rightarrow \text{Bag}(T_2) \]
\[ \phi_M(A, F) : T_2 \rightarrow \mathbb{N} \]
\[ \text{dom}(\phi_M(A, F)) = \bigcup_{j \in \text{dom}(A)} \text{dom}(F(j)) \]
\[ \phi_M(A, F)(i) = \sum_{j \in \text{dom}(A)} A(j) \cdot (F(j))(i). \]

When we consider vagueness we get the fold operator for vague multisets:

\[ \phi_{M, 3} : \widetilde{\text{Bag}}(T_1) \times (T_1 \rightarrow \widetilde{\text{Bag}}(T_2)) \rightarrow \widetilde{\text{Bag}}(T_2) \]
\[ \phi_{M, 3}(V, F_3) = X, \text{ with} \]
\[ \lambda_X(i) = \sum_{j \in T_1} \lambda_{F_3(j)}(i) \cdot \lambda_{F_3(j)}(i) \]
\[ v_X(i) = \sum_{j \in T_1} v_{F_3(j)}(i) \cdot v_{F_3(j)}(i) \]
\[ \Delta_X = \bigcup_{i \in \Delta_V} \Delta_{F_3(i)}. \]

Since the selection operator can be seen as an instantiation of the fold operator, the rules for the selection operator can be easily derived from the above rules.

7.3 Elimination of Duplicates

Almost every query language that operates on multisets offers a facility to eliminate duplicates. Formally speaking, there is an operation \( \varsigma \) with \( \varsigma : \text{Bag}(T) \rightarrow \text{Set}(T) \) and \( \varsigma(A) = \text{dom}(A) \). Taking vagueness into account, we get the vague variant \( \varsigma_3 : \text{Bag}(T) \rightarrow \text{Set}(T) \) of \( \varsigma \) with:

\[ \varsigma_3(V_i) = \{i \in \Delta_V \mid v_V(i) > 0\} \]
\[ \varsigma_3(V_a) = \Delta_V \]
\[ \delta_3(V_j)(i) = \begin{cases} t, & \text{if } v_V(i) > 0 \\ f, & \text{if } v_V(i) = 0 \\ u, & \text{otherwise.} \end{cases} \]

From the above considerations, the analogies between vague sets and vague multisets should have become obvious. Hence, our approach represents a closed calculus for vague sets and vague multisets.

8 IMPLEMENTATION ASPECTS

In this section, we treat some aspects of our realization of the concepts proposed in this paper. Our implementation is based upon H-PCTE [19], a high performance implementation of PCTE. It directly uses the Propagation Rules described in this paper. A vague set V is implemented as a triple consisting of 1) a set containing the elements in \( \hat{V}_i \), 2) a set containing the elements in \( \hat{V}_a \), and 3) a list of predicate trees representing the descriptive parts which arise when we trace back 1, 2, ..., i levels in the query defining the set, where i is the depth of the nesting of the query. Due to the maintenance of descriptive parts with different accuracies, it is possible to weight the expected computation costs against the expected benefit whenever applying the descriptive part.

In the implementation, the descriptive part is first of all used in the Propagation Rules for binary operators. In addition, it can be useful to consider the descriptive part of a query result carefully to see whether further improvements are possible with reasonable effort. If we know, for example, that elements of a concrete type might be missing in the result, and all objects of this type can be accessed efficiently, because there are only few of them which are physically clustered, it could be worthwhile to check all of them against the descriptive part of the vague result set. This can enhance the explicit part of the result set and thus, the expressiveness of the result for the user.

At the moment, our implementation does not include a sophisticated optimizer. However, such an optimizer seems to be an extremely important topic for further research activities, because there are many new requirements for the query optimizer in our environment. 1) The optimizer has to combine static and dynamic aspects, because depending on the accessibility situation observed during query processing different possibilities to continue can occur. 2) Cost
estimation and accuracy estimation functions have to be developed for different descriptive functions in order to choose the “optimal” back tracing level in a given situation. 3) Given a concrete accessibility situation, the optimizer should choose the query execution plan for which the ratio between the expected execution time and the expected inaccuracy is optimal.

Another point where the descriptive part of the result of a query can be useful is the generation of an expressive error report clarifying the degree of the possible incorrectness and the reasons. In this respect, especially, the consideration of locality information and information gained by the exploitation of fine-grained typing may be useful.

Finally, the available opportunities to build the descriptive parts depend on the underlying data model. In PCTE, for example, (almost) every link has a reverse link. Therefore, we can test the existence of an incoming link by the existence of its reverse link. Many object-oriented data models (e.g., the ODMG data model [3]) offer similar concepts. Furthermore, other data models—such as the relational data model—also offer interesting features with respect to our approach. For example, an efficient access to all elements of a given type can be exploited to improve the explicit parts of a vague set, as above explained.

9 Related Work

In the context of relational databases, much work has been done in order to deal with unknown and not completely known values. The general technique to deal with problems of this kind is null values [4]. This concept, introduced by Codd, was later generalized by himself and by other researchers.

One possible improvement is to distinguish different reasons for a value to be unavailable. Gessert [11], [12] and Vassiliou [27] propose the use of a four valued logic to distinguish “values not known” and “values not applicable.” The state “value is not applicable” can occur if data modeling was oversimplifying, and hence there are instances of a type that do not have all fields of this type.

Another means to express uncertainty for an individual value is disjunctive information [30]; it consists of a logical disjunction of several possible values of which one is certainly the correct one.

Additionally to disjunctive information, Liu and Sunderraman [23], consider so-called maybe information, i.e., information that might be correct. This approach is based upon the closed world assumption, assuming that all information not contained in the database is incorrect.

Lipski [21] proposes the concept of null variables; the idea is to use the same null variable for two values that are unknown but must have the same value.

A more sophisticated approach is to describe an unknown value (“value not known”) by a fuzzy set [6], [31], [32], i.e., by a set of values plus their probability of being the missing one. In [25], Motro surveys approaches dealing with the use of fuzzy sets for the processing of uncertain queries. Other articles dealing with fuzzy sets in the context of query results are, for example [2], [10], [17]. The use of fuzzy sets is only possible if probabilistic information is available, a prerequisite that is not fulfilled in our context.

All the above-mentioned concepts have one principle in common: The vagueness of the data that is stored in the database implies the vagueness of the result of the query.

A basic difference between incomplete values denote unknown attribute values, while in the case of inaccessibility even the existence of entire objects is unknown. This difference leads, for example, to different results when computing the cardinality of a set. Furthermore, missing objects cannot be replaced by objects consisting only of null values, because the number of missing objects is in general unknown, too.

Although the concepts for incomplete attribute values are not adequate to deal with vagueness in the context of partly inaccessible databases, the respective research is nevertheless interesting for us. For example, the computation of aggregate operations on vague sets may cause values that are only incompletely known [26]. Hence, partly inaccessible databases lead to the consequent problem of unknown values. To deal with this kind of vagueness, we can employ the techniques developed in the context of incomplete data.

Lipski [21] distinguishes the internal and the external interpretation of a query. The external one corresponds to the complete information about the real world. The internal one corresponds to a result that is given according to the actual incomplete information in the database. It is approximating the external interpretation by consisting of a lower and an upper bound. Because Lipski investigates missing information at the level of atomic attributes, he does not need to concern query language operations, such as selection, on the sets above described. He is only interested in the usual set operations. Besides, in his context the upper bound of a result set can always be stated explicitly, a fact which is not true in the case of inaccessibility in distributed systems.

The articles [24], [2], [1], [30] are all concerned with extensions of Lipski’s approach, for example, by adding statistical information. Because accessibility of the relevant data is always presupposed, there is no need for a hybrid representation.

Imielinski and Lipski [17] define conditions that must be fulfilled by the semantics of a relational algebra in order to allow the correct computation of a query involving unknown values. A query is computed correctly, in this sense, if all possibilities for the meaning of the unknown values are contained in the result. Imielinski and Lipski define so-called representation systems that consist of a subset of the usual relational algebra, the class of allowed unknown values, and the extension of the used relational operators to these unknown values. The representation
systems in turn guarantee the correct computation described. This work, however, strictly refers to the relational data model instead of an object oriented model, and it does not take inaccessibility into account; thus, there is no need to use a descriptive part for the missing items.

Read et al. [26] describe a technique to approximate the result of an aggregate operation over a set by representing this set as a so-called histogram. A histogram consists of a partition of the set and for each subset of the partition the number of elements of the origin set contained within the subset. The aim of this work is to save run time for the evaluation of an aggregate operation, indeed at the cost of precision. A histogram has not much in common with our understanding of a vague set; the common part is that a somehow “vague” representation of a set produces an only incomplete known value for an aggregate operation.

In the area of geographical information systems, the determination of the topological relation two regions are in is of interest [7]. In this context, different sources of imprecision, e.g., the incorrect evaluation of air or satellite photos [8], cause vague regions. Cohn and Gotts use the so-called “egg yolk” representation for geographical regions with undetermined boundaries [5]. This representation corresponds to a graphical impression of our vague sets. Because of their application area, however, Cohn and Gotts neither use a hybrid representation nor do they consider the query language operators considered in our paper.

10 Conclusion

In this paper, we have presented an approach to deal with vague collections resulting from inaccessibility in distributed object bases. We have shown that, in this context, a hybrid representation is needed that consists of 1) an enumerating part, which contains the elements we could access during query processing, and 2) a descriptive part, which describes the relevant elements we could not access. Further, we have introduced Propagation Rules which can be used to minimize the vagueness of a query result represented by the hybrid representation.

Whereas vague multisets can be handled quite similarly to vague sets, further research is needed with respect to vague lists, for example, resulting from sorting operations. Since the sorting criterion must also be considered three-valued, we have to deal with a vague order in this context.

A possibility to improve the descriptive part of the hybrid representation not yet considered in depth is exploiting the criteria used for clustering. Many object management systems enable the user to state these criteria explicitly as predicates. Knowing these predicates for the inaccessible parts of the object base, we can further restrict the missing elements.

Another interesting matter of research is the interaction of the query optimizer and the Propagation Rules. In our application scenario, the query optimizer has to consider the potential inaccuracy of the query result as a further optimization criterion.

References


Oliver Haase received the MS degree in computer science in 1993 from the University of Karlsruhe, Germany. In 1993, he began working as a research assistant with the group for practical computer science at the University of Siegen, Germany; his research focused mainly on distributed and semantic databases, query languages, and partial inaccessibility of databases. In 1997, funded by the German Research Community DFG, he received the PhD degree. Currently, he is a research staff member of the Computer and Communication Research Laboratories of NEC Europe Ltd. in Heidelberg, Germany. He is responsible for middleware technology, software architecture, and network intelligence.

Andreas Henrich received the MS degree in information systems in 1987 from the Darmstadt University of Technology, Germany. From 1987 to 1990, he was a research assistant at the University of Hagen, Germany, where he received the PhD degree in computer science for a thesis about spatial access structures. Thereafter, he worked as a systems analyst at Deutsche Bank AG in Frankfurt for three years. From 1993 to 1998, he was an assistant professor with the Group for Practical Computer Science at the University of Siegen, where he received the habilitation degree (venia legendi) for computer science in 1997. Currently, he is an associate professor for practical computer science on the Faculty of Social and Economic Sciences at the Otto-Friedrich University of Bamberg, Germany. His research interests include multidimensional access structures, information retrieval, query languages, and multimedia databases.