To Form a More Perfect Union (Intersection, Difference)

Mitch Cherniack*
Brown University
Providence, RI, USA
mfc@cs.brown.edu

Stanley B. Zdonik*
Brown University
Providence, RI, USA
sbz@cs.brown.edu

Marian H. Nodine*
Brown University
Providence, RI, USA
mhn@cs.brown.edu

Abstract

The AQUA [16] query algebra allows user-defined equivalence relations as arguments to query operators that generalize standard set operations. These predicates determine what objects are included in the query result, and the duplicates that must be removed.

While an expressive enhancement, the use of arbitrary equivalence relations to decide set membership can result in sets with counterintuitive behavior, and therefore can make queries return unreasonable results. In this paper, we show that equality predicates assume two roles with respect to sets. Distinguishers differentiate between set members and implicitly give meaning to standard set properties such as set equality. Constructors determine which object from input sets contribute to the query result. The requirements of distinguishers and constructors differ. AQUA's set operators are problematic because they use constructors where distinguishers are required. We propose alternatives to AQUA's set operators that address this limitation.

1 Introduction

There is something universal about equality. It is reasonable to write both $3 + 2 = 4 + 1$, and $\{1, 2\} = \{2, 1\}$, and the same conclusions can be drawn from these statements: the equal values are in fact singular and need not be referred to in the plural. That is, $\{1, 2\}$ and $\{2, 1\}$ are simply different ways to denote the same thing, and these denotations are freely substitutable for one another.

Universality breaks down when equality must be implemented as a decision-making algorithm (or *equality predicate* ($\equiv$)). Algorithms are dependent on representations, and because representations will differ according to type, equality implementations must also necessarily differ. Thus integer equality (which might be implemented as a bit-wise comparison of bit strings) must necessarily have a different implementation than set equality (which will likely be implemented using looping). For this reason, it is generally accepted ([3], [9], [11], [12], [16], [4]) that users should be able to provide their own user-defined equality predicates specific to user-defined types.

The query algebra, AQUA [16] takes this philosophy a step further. AQUA permits multiple equality predicates to be used for objects of a given type. This is motivated by the recognition that what makes two objects equal might not only be relative to their representation but also to how they are used. For example, a demographer might consider two people to be equal if they belong to the same age group (`agegp`). For sampling purposes, the demographer might use this notion of equality to query a set of people who are of different ages. On the other hand, a physician would likely want a finer-grained notion of when two people are equal (perhaps based on a social security number (`SSN`)) to ensure that only the right person is given a particular medical treatment. AQUA provides support for multiple equality predicates by making equality predicates parameters to queries. As the demographer and physician might access the same set of people, AQUA permits both users to pose queries relative to their particular interpretations of equality.

To illustrate, we show three sets of people and two possible equality predicates that compare people in Figure 1. The AQUA query, $A \cap_{\equiv_{dem}} B$ which intersects A and B relative to the equality predicate, $\equiv_{dem}$, can return either of

*Partial support for this work was provided by the Advanced Research Projects Agency under contract N00014-91-J-4052 ARPA order 8225, and contract DAAB-07-91-C-Q518 under subcontract F41100.

$$
\begin{array}{lcll}
A & = & \{ & \ll \texttt{ssn: } 1, \texttt{name: } \textit{Jack}, \quad \texttt{agegp: } \textit{30's} \gg, \\
 & & & \ll \texttt{ssn: } 2, \texttt{name: } \textit{Jill}, \quad \texttt{agegp: } \textit{20's} \gg \} \\
 \\
B & = & \{ & \ll \texttt{ssn: } 3, \texttt{name: } \textit{Jim}, \quad \texttt{agegp: } \textit{40's} \gg, \\
 & & & \ll \texttt{ssn: } 4, \texttt{name: } \textit{Ann}, \quad \texttt{agegp: } \textit{30's} \gg \} \\
 \\
C & = & \{ & \ll \texttt{ssn: } 1, \texttt{name: } \textit{Jack}, \quad \texttt{agegp: } \textit{30's} \gg, \\
 & & & \ll \texttt{ssn: } 6, \texttt{name: } \textit{Bill}, \quad \texttt{agegp: } \textit{20's} \gg \}
\end{array}
$$

$$
\begin{array}{lcl}
\equiv_{dem} & \stackrel{\text{def}}{=} & \lambda(x, y)\ x.\texttt{agegp} = y.\texttt{agegp} \\
\equiv_{phy} & \stackrel{\text{def}}{=} & \lambda(x, y)\ x.\texttt{ssn} = y.\texttt{ssn}
\end{array}
$$

Figure 1: Three sets of *Students* and potential user-defined equalities.

the sets $\{$*Jack*$\}$ or $\{$*Ann*$\}$. Both *Jack* and *Ann* belong to the intersection because *Jack* "equals" *Ann* according to $\simeq_{dem}$. But also, since *Jack* and *Ann* are "equal", only one of them belongs in the result.

Thus, set intersection is *well-defined* if we view well-definedness relative to the equality predicate used in determining the result. However, the potential results are distinguishable (we can distinguish them for example, by examining the `name` field of objects in the results) and therefore the effect of using a user-defined equality is to make set intersection potentially nondeterministic.

The flexibility attained by making equality predicates parameters to queries comes at a price. Complex queries that query on nondeterministic results of other queries may not be well-defined. For example, the complex query

$$(A \cap_{\equiv_{dem}} B) \cap_{\equiv_{phy}} C$$

will return $\{$*Jack*$\}$ if the intermediate result, $A \cap_{\equiv_{dem}} B$ returns $\{$*Jack*$\}$, but will return $\oslash$ if $A \cap_{\equiv_{dem}} B$ returns $\{$*Ann*$\}$. (This is because no student in C has the same `ssn` as *Ann*.) $\oslash$ is not equal to $\{$*Jack*$\}$ modulo $\equiv_{dem}$ or $\equiv_{phy}$, and the reasonableness of the result is lost.

In this paper, we present the reasons why AQUA set queries such as these are problematic and propose an alternative algebra that is in the same spirit but is deterministic, and hence well-defined. The crucial observation we make is that there are **two** roles that equalities can assume with respect to sets, and an equality predicate must be defined with its intended role in mind. Equality predicates that are used as *distinguishers* are permanently associated with sets and implicitly determine when an object belongs to a set and when two sets are equal. *Distinguisher* definitions must obey strict constraints to ensure proper set behavior. *Constructor* equalities are used to build new sets from existing sets, but are not associated with constructed sets after they are built. *Constructor* definitions permit more flexibility than do distinguishers. AQUA's set operators are problematic because they accept *constructors* as arguments, but use them as *distinguishers*.

The paper is organized as follows. In Section 2, we consider the differences between *distinguishers* and *constructors* and propose constraints on *distinguisher* definitions that ensure that sets and set operators behave as they should. In Section 3, we present our alternatives to AQUA's set operators, and show that our operators are well-defined and satisfy expected identities such as commutativity, associativity, idempotence and deMorgan laws. In Section 4, we present related work and contrast our approach to others. In Section 5, we summarize directions for future work.

2 The Role of User-Defined Equalities in Sets

We assume a universe of typed objects, where a type defines the *attributes* of an object. An attribute m of an object o is accessed using "dot notation", as in $o.m$.

An attribute is *mutable* if it's value can be changed. An object (type) is mutable if it contains (defines) a mutable attribute. An object (type) is *immutable* if it is not mutable.

Mutable objects do not lose their identity when they are mutated. Thus, we assume that the underlying system provides support for *object identifiers* for mutable objects. Object identifiers are assumed to be known only to the system, and are associated with objects throughout their lifetimes. (Typically, object identifiers are implemented as pointers (as in the OID's of [15]) although we impose no restrictions on their implementation here.) Two objects with the same object identifier are indistinguishable, even via mutation, and hence are referred to as *identical*.

We assume the existence of *base* (system-provided) types such as integers, bools and immutable, homogeneous sets. All other types must be defined by the user. Equality for base types is defined in the usual way (e.g., based on *extensionality* (memberwise equality) for sets). Equality for non-base types must be explicitly defined by way of user-defined equality predicates. Equality predicates for mutable types can be defined to compare object identifiers but need not.

Equality predicates introduce a tension between the expressivity of a query algebra and the proper behavior of sets. Arbitrary equivalence relations used as equality predicates make it possible (as shown in Section 1) to succinctly express a wide variety of queries over sets. But most equivalence relations, when used to distinguish the members of a set, make sets and set operators behave in counterintuitive ways. In this section, we explore this tension and conclude that there are two roles that equality predicates can assume:

- *Distinguisher equalities* (hereafter refered to by $\simeq$) are permanently associated with sets and are used to distinguish its members (i.e, duplication is relative to *distinguisher* equalities). User-defined predicates that are used as distinguishers must satisfy restrictive properties to ensure that the sets for which they distinguish members are well-behaved.

- *Constructor equalities ($\equiv$)* are used to decide how new sets are constructed from existing sets (e.g., a *constructor* would decide if objects from two sets A and B are equal, and therefore if they belong in the intersection, $A \cap B$). *Constructors* are used once to construct a set and are not needed thereafter. Because of their impermanence, their definitions need not be as constrained as *distinguisher* equalities to ensure reasonable set behavior.

Distinguisher and *constructor* equalities should **not** be constrained to be the same. This division of responsibility facilitates a balance of expressivity and well-behavedness. Further, the use of *constructors* as explicit parameters to set operators need not introduce nondeterminism into the algebra. A deterministic algebra ensures that well-definedness problems of the kind described in Section 1 are avoided.

2.1 Distinguisher Equalities

Every set must have a *distinguisher* equality that differentiates its members. Standard set theory assumes this predicate as given and definitions for set concepts such as cardinality and extensionality depend on its existence. In this section we present generalizations of set concepts that account for *distinguisher* equalities that could be user-defined.

2.1.1 Set Theory Relative to User-Defined Equality Predicates

In Figure 2, we define properties of sets that depend on *distinguisher* equalities associated with set members. These properties include *membership* ($\in_{\simeq}$), *equality of sets* ($\simeq_{\{\}}$) and *cardinality* ($|\ |_{\simeq}$). For the axioms shown, we adopt the algebraic set notation used in Larch [13] that includes *insert* and $\oslash$ as set constructors. This is in contrast to the polyadic brace notation, $\{\ \ldots\ \}$, used in [5] for example.[1]

The pervasiveness of *distinguisher* equality predicates in these definitions helps argue our point that properties held or not held by $\simeq$ affect the well-behavedness of the set for which it distinguishes members. Below we describe some unreasonable set behaviors that can result from poorly defined *distinguishers*. We describe each behavior and show how poor *distinguisher* definitions are at fault. We then propose constraints on *distinguisher* definitions that guard against these unintuitive behaviors.

[1] Our notation is consistent with the assumption that sets are immutable because *insert* is an algebraic constructor and not a mutator. That is, the set $insert\ (2,\ insert\ (1,\ \oslash))$ denotes a different set than $insert\ (1,\ \oslash)$, and is not a mutation of this set by insertion.

$$\begin{array}{lll} e \notin_{\simeq} \oslash & & \\ e \in_{\simeq} insert(e', S) & \Longleftrightarrow & (e \simeq e') \vee (e \in_{\simeq} S) \end{array}$$

Set membership

$$S \simeq_{\{\}} T \quad \Longleftrightarrow \quad \forall e \; (e \in_{\simeq} S \Leftrightarrow e \in_{\simeq} T)$$

Equality of sets (*extensionality*)

$$\begin{array}{lll} |\oslash|_{\simeq} & = & 0 \\ |insert(e, S)|_{\simeq} & = & \textit{if } e \in_{\simeq} S \textit{ then } |S|_{\simeq} \textit{ else } 1 + |S|_{\simeq} \end{array}$$

Cardinality

Figure 2: Some $\simeq$-Parameterized Axioms for Sets

2.1.2 Symptom 1: Equal Sets that are Distinguishable by Querying

In CLU [17] it is argued that equality definitions should be *congruences*; two objects should be equal only if they are *indistinguishable*. (Two objects are distinguished if they return different values for the same attribute.) This should be no different for sets: equality of sets should establish two sets to be indistinguishable by the query operators invokable on sets. This is because the equality of two sets makes them equally valid results to a given query. If the sets are distinguishable, the query is nondeterministic. Complex queries, which query over the result of the nondeterministic query, are sure to produce unequal results. (This was the case for the problematic example shown in Section 1). Therefore, equal sets must be indistinguishable to ensure that the queries that produce them as results of subqueries are well-defined.

For set equality (extensionality) to be a congruence, the *distinguisher* definition that is used to compare set members must be a congruence also. A *distinguisher* that is not a congruence can lead to elements that are deemed equal but that can be distinguished according to some attribute. Sets deemed equal can then be differentiated by then selecting or projecting over this attribute. (For example, the $\simeq_{dem}$-equal sets A and C of Figure 1 can be distinguished by projecting the `name` attributes ($\pi_{name}\ (A) \not\simeq_{\{\}} \pi_{name}\ (B)$).

2.1.3 Symptom 2: Equal Sets that have Different Cardinalities

Consider a *distinguisher* definition, $\simeq$ that is not symmetric. For example, suppose for objects a and b that

- $a \simeq a$ and $b \simeq b$,
- $a \simeq b$, but
- $b \not\simeq a$.

In this case, the sets below are equal by extensionality (by the definition of $\simeq_{\{\}}$ of Figure 2):

$$insert(a, insert(b, \oslash)) \quad \simeq_{\{\}} \quad insert(b, insert(a, \oslash)),$$

but have unequal cardinality according to the cardinality definition of the same figure. The cardinality of the set on the left is:

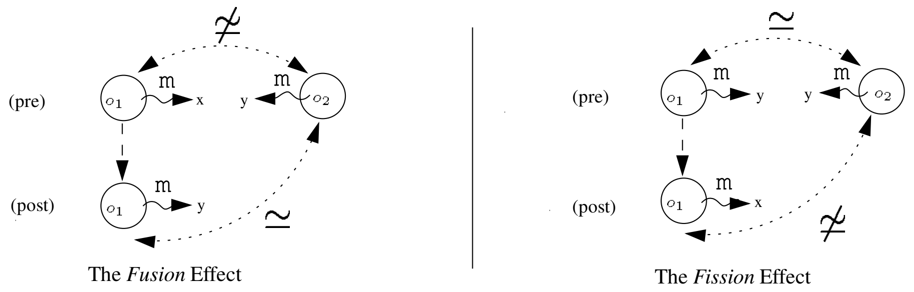

Figure 3: The *Fusion* and *Fission* Effects

$$\begin{aligned} |insert(a, insert(b, \oslash))|_{\simeq} &= |insert(b, \oslash)|_{\simeq} \\ &= 1 \end{aligned}$$

and the cardinality of the set on the right is:

$$\begin{aligned} |insert(b, insert(a, \oslash))|_{\simeq} &= 1 + |insert(a, \oslash)|_{\simeq} \\ &= 2 \end{aligned}$$

That equal sets have different cardinalities is unreasonable, and is a symptom of distinguishers that are not equivalence relations. The example above shows that distinguishers over set members must be *symmetric*. (If $b \simeq a$, both sets above have cardinality 1). Similarly, if $\simeq$ is not *reflexive* (e.g., $a \not\simeq a$), the extensionally equal sets

$$insert(a, insert(a, \oslash)) \quad \simeq_{\{\}} \quad insert(a, \oslash)$$

have cardinalities 2 and 1 respectively, while for a *distinguisher* $\simeq$ that is not *transitive* ($a \simeq b$ and $b \simeq c$ but $a \not\simeq c$), the extensionally equal sets

$$insert(a, insert(b, insert(c, \oslash))) \quad \simeq_{\{\}} \quad insert(a, insert(c, insert(b, \oslash)))$$

have cardinalities 1 and 2 respectively.

2.1.4 Symptom 3: Immutable Sets with Variable Cardinalities

The cardinality of an immutable set should be invariant. An immutable set does not have mutators that add or remove elements. However, the cardinality of an immutable set of mutable objects can change as a side-effect of the mutation of a member object if the *distinguisher* for this set is poorly defined. Changes in immutable set cardinality are counterintuitive and can impose a performance overhead on a system, as tests for duplicates in a set are then required after mutations of set members.

Variations in set cardinality come in two flavors. *Fusion* occurs when the mutation of an object results in its becoming equal to another object in a set, thereby decreasing set cardinality. *Fission* occurs when mutation results in two equal objects becoming unequal, thereby increasing set cardinality. Figure 3 illustrate both phenomena. In this figure, objects are denoted by circles, `m` is an object attribute that is mutable, and object o_1 is shown before (*pre*) and after (*post*) a mutating operation changes its value for `m`. The *distinguisher* assumed uses the value for `m` in its comparison, and therefore the mutation of o_1 can determine whether or not it is equal to o_2.

A *distinguisher* is *persistent* if it decides whether or not two objects are equal independently of their mutable state. A *distinguisher* that is persistent can ensure that the set it serves has invariant cardinality. The *distinguisher* used in Figure 3 is not persistent as it is dependent on the values returned by the mutable attribute `m`.

2.1.5 Suggested Restrictions on Distinguisher Definitions

We have shown that a *distinguisher* must be:

- a congruence,
- an equivalence relation and
- persistent

to ensure that the sets whose members are compared using the distinguisher are well-behaved. These properties are independent of one another. A congruence need not be an equivalence relation; $\lambda(x, y)\ false$ is trivially a congruence but is not reflexive. A congruence might not be persistent as a congruence does not guarantee that two unequal objects do not become equal.

These properties are undecidable in general, but can be ensured by constraining *distinguisher* definitions. For example, one can ensure that a *distinguisher* for objects of some type T is an equivalence relation by constraining its definition to be of the form,

$$\simeq \ \stackrel{\text{def}}{=} \ \lambda(x, y)\ x.\mathtt{m}_1 \simeq_1 y.\mathtt{m}_1 \ \wedge \ \ldots \ \wedge \ x.\mathtt{m}_\mathtt{n} \simeq_n y.\mathtt{m}_\mathtt{n}$$

where each $\mathtt{m_i}$ is an attribute defined over T, and each "invoked" equality predicate, $\simeq_i$ is a base type equality or a user-defined equality predicate that has no circular dependence on T . If each $\mathtt{m_i}$ is constrained to be an immutable attribute, then $\simeq$ will be persistent. If together, the attributes $\mathtt{m}_1 \ldots \mathtt{m}_\mathtt{n}$ comprise a key, then $\simeq$ will be a congruence. Distinguisher definitions that are of this form, and that also ensure that identical mutable objects are considered equal are hereafter referred to as *well-behaved distinguishers*.

The above constraints on *distinguisher* definitions guarantee proper set behavior but are restrictive, as we show below.

Lemma 2.1 (Type-Uniqueness of Distinguishers for Mutable Objects) *Given some type of mutable objects, T, T objects a and b, and well-behaved distinguisher for T-objects, $\simeq$, then $a \simeq b \Rightarrow a$ and b are identical.*

Proof: The proof is by contradiction: assume $a \simeq b$, but that a and b are not identical. Let m be a mutable attribute of T (m must exist as T is mutable.). Either m is one of the attributes of T objects compared within $\simeq$ or it is not. If it is, then $\simeq$ cannot be persistent as it is possible to change the m attribute of a without doing so for b, making it possible for a and b to "become" equal or unequal. If it is not, then $\simeq$ cannot be a congruence as it is possible for a and b to be equal despite having unequal values for m. As $\simeq$ has been assumed to be well-behaved, we have a contradiction in either case and the lemma is proved. □

Theorem 2.1 (Type-Uniqueness of Distinguishers) *Given some type T, T objects a and b, and (well-behaved) distinguishers for T-objects, $\simeq$ and $\simeq$',*

$$a \simeq b \ \Leftrightarrow \ a \simeq' b.$$

Proof: If T is an immutable type, then $a \simeq b$ implies that a and b are indistinguishable (by congruence). Therefore, any other distinguisher $\simeq'$ will decide that a and b are equal also, as $\simeq'$ will involve a comparison of attributes that were insufficient to distinguish a and b. Thus, $a \simeq b \ \Rightarrow \ a \simeq' b$. Similar reasoning establishes that $a \simeq' b \ \Rightarrow \ a \simeq b$.

If T is a mutable type, then it must be the case that $a \simeq b$ implies that a is identical to b (by **Lemma 2.1**), which in turn implies that $a \simeq' b$ (as identical objects are deemed equal by well-defined distinguishers.) Similar reasoning establishes that $a \simeq' b \ \Rightarrow \ a \simeq b$. □

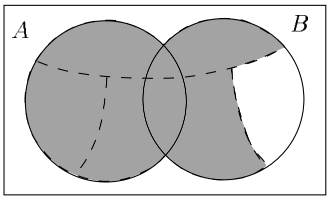

Figure 4: $A \uplus_{\equiv} B$ = A $\cup$ $B^{A}_{\equiv}$

2.2 Constructor Equalities

A *distinguisher* underlies a set throughout the lifetime of the set and therefore a *distinguisher* definition should be constrained to ensure proper set behavior. On the other hand, a *constructor* is only associated with a set upon its creation, and therefore a *constructor* definition need only be constrained to be an equivalence relation.

AQUA's set operators accept *constructors* but use them to eliminate duplicates from results. (In other words, they use *constructors* as *distinguishers*.) In the next section, we describe how AQUA's operators can be replaced with operators that accept *constructors* as inputs and that use the supplied *constructors* to determine which objects from input sets contribute to the result. Our operators differ from AQUA's operators in that the supplied *constructor* is not used to remove duplicates. Rather, the *distinguisher* for the resulting set is the same as the *distinguisher* for the input sets and is determined by the type of the object contained in those sets.

3 An Equivalence-Parameterized Set Algebra

Our operators are generalized forms of set union, intersection and difference. All of our operators are defined in terms of the operator, *extend* ($\uplus_{\equiv}$) which is parameterized by the *constructor*, $\equiv$. This operator, though *not* part of our proposed set algebra, simplifies its presentation.

$A \uplus_{\equiv} B$ is a superset of A whose members are "extended" to include members of B that are $\equiv$-related to members of A. We begin by formally defining this operator, and then defining our set operators, $\cup_{\equiv}$, $\cap_{\equiv}$ and $-_{\equiv}$ in terms of $\uplus_{\equiv}$. Finally, we present proofs of algebraic identities that hold of these operators. These proofs were verified with LP; the theorem prover of Larch [13].

3.1 The *Extend* Operator ($\uplus_{\equiv}$)

We define the extend operator over sets in terms of a filtering operator, $B^{A}_{\equiv}$.[2] For any two sets A and B, $B^{A}_{\equiv}$ consists of all elements in the set B that are equivalent to some element of A. More precisely, given a type T with distinguisher $\simeq$ and sets of T's, A and B:

$$B^{A}_{\equiv} \stackrel{\text{def}}{=} \{b \in_{\simeq} B \mid \exists a \, (a \in_{\simeq} A \ \wedge \ b \equiv a)\}$$

Note that by Theorem 2.1, it makes no difference which well-behaved distinguisher for T is chosen (as all will agree on what objects are equal), and therefore the above definition is well-defined.

The *extension* of A with respect to B includes elements from both A and this set. That is, $A \uplus_{\equiv} B$ = (A $\cup$ $B^{A}_{\equiv}$) (where $\cup$, $\cap$ and $-$ are defined in standard fashion with respect to an underlying distinguisher equality). This set is illustrated in Figure 4. We use Venn notation, with solid circles indicating set membership (with respect to distinguisher equalities), and dotted lines designating partitions induced by the equivalence relation $\equiv$. Shading denotes the contents of the extension set.

[2] $B^{A}_{\equiv}$ is read "B filtered by A with $\equiv$".

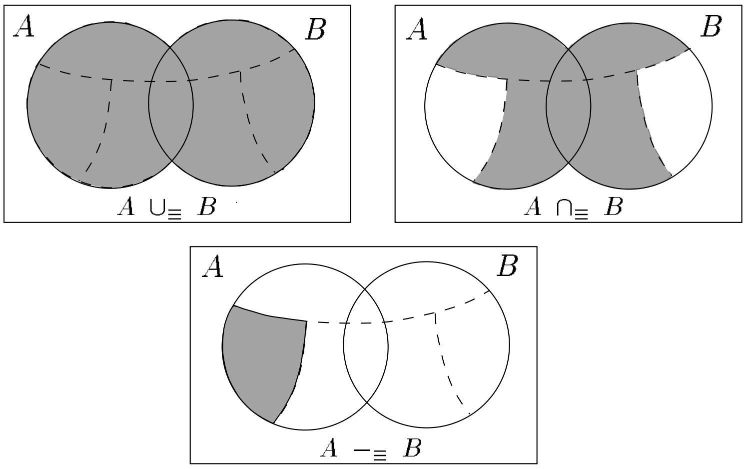

Figure 5: $\cup_\equiv$, $\cap_\equiv$ and $-_\equiv$ illustrated

3.2 Generalized Union, Intersection and Difference

The other operators in our set algebra are defined with respect to *extensions*. For any set operator $\circ$, the meaning of the equivalence-parameterized expression $A \circ_\equiv B$ is $(A \uplus_\equiv B) \circ (B \uplus_\equiv A)$. Below we give alternative definitions for generalized set operators with their provably equivalent simple forms:

$$
\begin{array}{lcllcl}
A \cup_\equiv B & \stackrel{\text{def}}{=} & (A \uplus_\equiv B) \cup (B \uplus_\equiv A) & = & (A \cup B) \\
A \cap_\equiv B & \stackrel{\text{def}}{=} & (A \uplus_\equiv B) \cap (B \uplus_\equiv A) & = & (A^B_\equiv \cup B^A_\equiv) \\
A -_\equiv B & \stackrel{\text{def}}{=} & (A \uplus_\equiv B) - (B \uplus_\equiv A) & = & (A - A^B_\equiv)
\end{array}
$$

These definitions are illustrated with Venn diagrams in Figure 5.

Unlike the equality-parameterized operations of [16], our operations are deterministic. One can get the effect of AQUA's nondeterministic operators by applying a representative choosing operation (such as AQUA's `dup_elim`) over any set resulting from the application of one of our operators. This is because our operators have the same functionality as those of [16] except that ours incorporate entire equivalence classes into the result rather than choosing representatives. Of course, if representatives are chosen in subqueries, then the same kinds of problematic behaviors described in Section 1 can be reintroduced. But most queries should not need to choose representatives until the last step. By making our set operators deterministic, we enhance the expressive power of the query algebra by allowing set operators to appear in subqueries without compromising the well-definedness of the query's result.

3.3 Identities in Our Algebra

We used LP ([13]) to prove a number of identities for the operators of Section 3.2. The proofs demonstrate that our set algebra generalizes standard set theory because the instantiation of our operators with distinguisher equalities results in functions with the semantics of $\cup$, $\cap$ and $-$. Also, our proofs show (when our operators are instantiated with any equivalence relation $\equiv$) that the following properties hold:

- $\cup_\equiv$ and $\cap_\equiv$ are *commutative*, *associative* and *idempotent*.
- $\cap_\equiv$ distributes over $\cup_\equiv$. Interestingly enough, it is not the case in general that $\cup_\equiv$ distributes over $\cap_\equiv$. This asymmetric distributivity is reminiscent of the arithmetic operators $+$ and $\cdot$.
- DeMorgan's Laws hold of $-_\equiv$ with respect to both $\cup_\equiv$ and $\cap_\equiv$.

$$
\begin{array}{rclrcl}
A -_{\equiv} B & = & A -_{\equiv} (A \cap_{\equiv} B) & A^{B}_{\cong} & = & A \cap B \\
(A \cup B)^{C}_{\equiv} & = & A^{C}_{\equiv} \cup B^{C}_{\equiv} & (A \cap B)^{C}_{\equiv} & = & A^{C}_{\equiv} \cap B^{C}_{\equiv} \\
(A - B)^{C}_{\equiv} & = & A^{C}_{\equiv} - B^{C}_{\equiv} & A^{(B \cup C)}_{\equiv} & = & A^{B}_{\equiv} \cup A^{C}_{\equiv}
\end{array}
$$

$$
\begin{array}{rclcl}
A \cap B & = & (A^{B}_{\equiv} \cap B^{A}_{\equiv}) & & \\
A^{(B^{C}_{\equiv})}_{\equiv} & = & (A^{B}_{\equiv})^{C}_{\equiv} & = & A^{B}_{\equiv} \cap A^{C}_{\equiv}
\end{array}
$$

Figure 6: Some Proven Identities in our Algebra

We also prove other identities that demonstrate the way in which these operators interact. Some of these are shown in Figure 6 (where A and B are arbitrary sets and $\equiv$ is an equivalence relation). Among the identities still open are closed forms for $A^{(B \cap C)}_{\equiv}$ and $A^{(B - C)}_{\equiv}$.

3.4 A Return to Our Motivating Example

It is worth reconsidering our motivating examples from Figure 1 with respect to the deterministic operators we propose. Given our operators, we have the following query results:

$$
\begin{array}{rcl}
A \cap_{\equiv_{dem}} B & = & \{\mathit{Jack}, \mathit{Ann}\} \\
A \cap_{\equiv_{phy}} B & = & \oslash \\
(A \cap_{\equiv_{dem}} B) \cap_{\equiv_{dem}} C & = & \{\mathit{Jack}, \mathit{Ann}\} \\
(A \cap_{\equiv_{dem}} B) \cap_{\equiv_{phy}} C & = & \{\mathit{Jack}\} \\
(A \cap_{\equiv_{phy}} B) \cap_{\equiv_{dem}} C & = & \oslash \\
(A \cap_{\equiv_{phy}} B) \cap_{\equiv_{phy}} C & = & \oslash
\end{array}
$$

We also have as results using the other set operators,

$$
\begin{array}{rclcl}
A \cup_{\equiv_{dem}} B & = & A \cup_{\equiv_{phy}} B & = & \{\mathit{Jack}, \mathit{Jill}, \mathit{Jim}, \mathit{Ann}\} \\
A -_{\equiv_{dem}} B & = & \{\mathit{Jill}\} & & \\
A -_{\equiv_{phy}} B & = & \{\mathit{Jack}, \mathit{Jill}\} & & \\
A -_{\equiv_{dem}} C & = & \oslash & & \\
A -_{\equiv_{phy}} C & = & \{\mathit{Jill}\} & &
\end{array}
$$

4 Related Work

Work on user-defined equality predicates is not new. Our contributions concern recognizing the dual roles that user-defined equality predicates play with respect to sets, and building an equivalence-parameterized and deterministic set algebra. Our work was motivated by recognition of the well-definedness problems of AQUA's nondeterministic operators ([16]) that were described in Section 1.

Others ([3], [9], [20]) have considered the role of user-defined equalities in query operators. In [3] and [9], sets are considered tied to a single notion of equality (equivalent in spirit to our distinguisher equality). Constructor equalities are not differentiated.

Our discussion of the duality of equality predicates most resembles observations made by Osborn in [20]. However, her perspective is inspired by querying. She writes that every object should have an identity that can be used for navigational querying as well as an equality that can be used for associative access, but fixes both predicates to be system-defined. We recognize that there may be any of a number of equivalences that are appropriate for use for associative access and that not every object should have identity (e.g., rational numbers should be differentiated by their values). [21] compares equality predicates but by their structure (the depth to which objects are compared) rather than their purpose.

Our thoughts on the requirements on distinguisher equalities resembles work done in [17], [12], [11], [4], [14], [1]. From CLU ([17]) we borrow the idea that two objects should be equal if and only if they are indistinguishable. This idea is carried further in the object-oriented setting in [12] and [11] which argue that equality definitions should vary according to the mutability of an object, and that equality for mutable objects should always depend on comparisons of identity. Baker [4] proposes that equal mutable objects should share side-effects. This differs from congruence in that the effects of mutating *one* of two equal objects is seen from the other. Guaranteeing shared side-effects for equality predicates is one way to prevent the *fission* effect. However, since our notion of persistence also establishes that unequal objects do not become equal (i.e., no *fusion*), it is a tighter constraint on equality definitions. Kosky [14] addresses the issue of when two database instances are equal, but does not consider when two objects belonging to the same database instance are equal. Abiteboul and Van den Bussche [1] consider object equality but ignoring how object mutability impacts equality predicate definitions.

Finally, [2] and [10] also consider nondeterminism in database query operators. Abiteboul et. al. [2] show that nondeterminism in query operators allows expression of certain polynomial-time (counting) queries as well as opportunities for optimization. We borrow our notion of what makes a nondeterministic operator well-defined from [10].

5 Conclusions and Further Work

This paper has proposed a separation between two fundamental uses of equality: distinguishing and constructing new sets. We have shown that three undesirable behaviors of sets can arise if the *distinguisher* that is used to compare set members is too weak. Specifically,

- equal sets can be distinguishable by queries if the *distinguisher* is not a *congruence*,
- equal sets can have different cardinalities if the *distinguisher* is not an *equivalence relation*, and
- the cardinality of immutable sets can vary over time if the distinguisher is not *persistent*.

We also have shown that *constructor* definitions need not be as tightly constrained as *distinguishers*, but that AQUA's set operators that accept *constructors* as arguments have well-definedness problems because they use *constructors* as if they were *distinguishers*. We solve this problem by proposing alternative set operators that also are parameterized by *constructors*, but that are deterministic (and therefore well-defined) and that preserve standard identities expected of operators over sets.

We would like to examine how our set algebra might be used to construct heterogeneous sets. For example, in a federated database it may be useful to combine two sets of people from different databases (and perhaps with differing representations). Our operators may be useful here, since the constructor equality used to combine the sets could compare just attributes that are common to the differently-typed objects. The issue that this introduces is what would be the distinguisher equality of the set that resulted? We could ask the user to supply a distinguisher equality when posing the query, or we could automatically construct an equality predicate that combined the distinguisher equalities of the sets that were input to the query.

The work described here is part of the development of a general-purpose query algebra that is independent of underlying data models. Our algebra, KOLA [8] is combinator-based, and includes the set operators described here as well as other operators described in [6, 8]. A Larch formal specification of KOLA, and theorem prover scripts that verify over 300 identities, including those described in Section 3, are described in [6]. We are currently concentrating our efforts on integrating KOLA within the Thor [18] object server under development at MIT and the EPOQ [19] extensible optimizer under development at Brown.

6 Acknowledgements

Special thanks are due to Barbara Liskov and Andrew Myers who inspired our work with thought-provoking questions about what it means for an equality predicate to be well-behaved. We thank Scott Vandenberg, Val Tannen and the reading group of the Oregon Graduate Institute who each gave extremely helpful comments on earlier drafts of this paper. We also thank Catriel Beeri, Peter Grogono, Gail Mitchell, Hagit Shatkay and Scott Vandenberg for sharing their ideas about equality with us.

References

[1] Serge Abiteboul and Jan Van den Bussche. Deep equality revisited. In Alberto O. Mendelzon Tok Wang Ling and Laurent Vieille, editors, *Deductive and Object-Oriented Databases, Fourth International Conference, DOOD'95*, Singapore, December 1995.

[2] Serge Abiteboul, Eric Simon, and Victor Vianu. Non-deterministic languages to express deterministic transformations. In *Proceedings of the Ninth ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 218–229, Nashville, Tennesee, April 1990.

[3] M. P. Atkinson, C. Lecluse, P. Philbrow, and P. Richard. Design issues in a map language. In Paris Kanellakis and Joachim W. Schmidt, editors, *Bulk Types & Persistent Data: The Third International Workshop on Database Programming Languages*, pages 20–32, Nafplion, Greece, August 1991. Morgan Kaufmann Publishers, Inc.

[4] Henry Baker. Equal rights for functional objects or, the more things change, the more they are the same. *OOPS Messenger*, 4(4):1–26, October 1993.

[5] Catriel Beeri and Yoram Kornatzky. Algebraic optimization of object-oriented query languages. In S. Abiteboul and P. C. Kanellakis, editors, *Proceedings of the Third International Conference on Database Theory*, number 470 in Lecture Notes in Computer Science, pages 72–88, Paris, France, December 1990. EATCS, Springer-Verlag.

[6] Mitch Cherniack. Form(ers) over function(s): The KOLA query algebra. Technical report, Brown University Department of Computer Science, May 1995. In preparation.

[7] Mitch Cherniack and Stanley B. Zdonik. Applying formal methods to the design of a query optimizer. Technical Report CS-95-41, Brown University Department of Computer Science, October 1995.

[8] Mitch Cherniack and Stanley B. Zdonik. Rule languages and internal algebras for rule-based optimizers. In *Proc. ACM SIGMOD Int'l Conference on Management of Data*, Montréal, Québec, Canada, June 1996.

[9] Richard Connor, Malcolm Atkinson, Sonia Berman, Quinton Cutts, Graham Kirby, and Ron Morrison. The joy of sets. In *Proc. 4th Int'l Workshop on Database Programming Languages*, pages 417–433, New York, New York, August 1993. Springer-Verlag.

[10] Jan Van den Bussche and Dirk Van Gucht. Semi-determinism (extended abstract). In *Proceedings of the Eleventh ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, 1992.

[11] Peter Grogono and Patrice Chalin. Copying, sharing and aliasing. In *Colloquium on Object-Orientation in Databases and Software Engineering*, Montreal, Quebec, 1994.

[12] Peter Grogono and Philip Santas. Equality and assignment in object-oriented languages. In *EastEurOOPe'93*, pages 191–201, November 1993.

[13] J.V. Guttag, J.J. Hornung, S.J. Garland, K.D. Jones, A. Modet, and J.M. Wing. *Larch: Languages and Tools for Formal Specifications*. Springer-Verlag, 1992.

[14] Anthony Kosky. Observational distinguishability of databases with object identity. In *Proc. 5th Int'l Workshop on Database Programming Languages*, Gubbio, Italy, September 1995. Springer-Verlag.

[15] Christophe Lecluse, Philippe Richard, and Fernando Velez. O_2, an object oriented data model. In *Proceedings of the Conference on the Management of Data*, pages 424–433, June 1988.

[16] Theodore W. Leung, Gail Mitchell, Bharathi Subramanian, Bennet Vance, Scott L. Vandenberg, and Stanley B. Zdonik. The AQUA data model and algebra. In *Proc. 4th Int'l Workshop on Database Programming Languages*, New York, New York, August 1993. Springer-Verlag.

[17] Barbara Liskov and John Guttag. *Abstraction and Specification in Program Development*. MIT Press, 1986.

[18] Barbara Liskov, Paul Johnson, Robert Gruber, and Liuba Shrira. A highly available object repository for use in a heterogeneous distributed system. In Alan Dearle, Gail M. Shaw, and Stanley B. Zdonik, editors, *Implementing Persistent Object Bases, Principles and Practice, Proceedings of the Fourth International Workshop on Persistent Objects*, Martha's Vineyard, MA, USA, September 1990. Morgan-Kaufman.

[19] Marian H. Nodine, Farah B. Abbas, and Mitch Cherniack. The EPOQ query optimizer for object-oriented databases: Design specification, 1994.

[20] S. L. Osborn. Identity, equality and query optimization. In K. R. Dittrich, editor, *Advances in Object-Oriented Database Systems:Proceedings of the 2nd International Workshop on Object-Oriented Database Systems*, number 334 in Lecture Notes in Computer Science, pages 346–351, Bad M¨unster am Stein-Ebenburg, FRG, September 1988. Springer-Verlag.

[21] Gail M. Shaw and Stanley B. Zdonik. Object-oriented queries: Equivalence and optimization. In Won Kim, Jean-Marie Nicolas, and Shojiro Nishioe, editors, *Proceedings of the The First International Conference on Deductive and Object-Oriented Databases*, pages 264–278, Kyoto, Japan, December 1989.