SEMANTIC REPRESENTATIONS AND QUERY LANGUAGES FOR OR-SETS*

Leonid Libkin
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA
Email: `libkin@research.att.com`

Limsoon Wong
Real World Computing Partnership Novel Function
Institute of Systems Science Laboratory
Heng Mui Keng Terrace, Singapore 0511
Email: `limsoon@iss.nus.sg`

Abstract

Or-sets were introduced by Imielinski, Naqvi, and Vadaparty for dealing with limited forms of disjunctive information in database queries. Independently, Rounds used a similar notion for representing disjunctive and conjunctive information in the context of situation theory. In this paper we formulate a query language with adequate expressive power for or-sets. Using the notion of normalization of or-sets, queries at the "structural" and "conceptual" levels are distinguished. Losslessness of normalization is established for a large class of queries. We obtain upper bounds for the cost of normalization. An approach related to that of Rounds is used to provide semantics for or-sets. We also treat or-sets in the context of partial information in databases.

*Appeared in *Journal of Computer and System Sciences*, 52(1):125–142, February 1996

1 Introduction

Applications within design, planning, and scheduling areas have motivated Imielinski, Naqvi, and Vadaparty to introduce the notion of *or-set* [17, 18]. Although or-sets are in essence disjunctive information, they are distinguished from the latter by having two distinct interpretations. An or-set can either be treated at a *structural* level or at a *conceptual* level. The structural level concerns the precise way in which an or-set is represented. The conceptual level sees an or-set as denoting an object which is equal to a member of the or-set. For example, the or-set $\langle 1, 2, 3\rangle$ is structurally a collection of numbers; however it is conceptually a number that is either 1, 2, or 3. (In this paper angle brackets $\langle\rangle$ are used for or-sets and $\{\}$ for the usual sets.)

The two views of or-sets are complementary. Consider a design template used by an engineer. The template may indicate that component A can be built by either module B or module C. Such a template, as explained in [17], is structurally a complex object whose component A is the or-set containing B and C. A designer employing such a template should be allowed to query the structure of the template, for example, by asking what are the choices for component A. On the other hand, the designer should also be allowed to query about possible completed designs, for example, by asking if there is a low cost completed design. In the latter case, as the designer is still in the process of creating a design, the "completed design" is purely conceptual. Both views of or-sets are important and should be supported.

The structural interpretation of or-sets is quite clear. However, the conceptual interpretation requires further exposition. A few operators at the structural level prescribing the interaction of or-sets, products and ordinary sets are needed for this purpose. These operators are used to express transformations among objects that are conceptually equivalent. Their goal is to transform any object X with or-sets into an or-set $\langle x_1, \ldots, x_n\rangle$, where x_is are the objects denoted by X, and x_is do not use or-sets. In other words, $\langle x_1, \ldots, x_n\rangle$ may be considered as the value that is represented by X, and the transformation $X \to \langle x_1, \ldots, x_n\rangle$ as the passage from the structural to the conceptual level. We shall see in Section 3 that the operators below are the only crucial ones for performing this transformation.

The operator $or\_\mu^s : \langle\langle s\rangle\rangle \to \langle s\rangle$ flattens an or-set of or-sets of type s. For example, applying or_μ to $\langle\langle 1, 2, 3\rangle, \langle 2, 4\rangle\rangle$ produces the or-set $\langle 1, 2, 3, 4\rangle$. The most important thing to note here is that *or_μ preserves the conceptual value of the input.* First $\langle 1, 2, 3\rangle$ is conceptually either 1, or 2, or 3. Similarly, $\langle 2, 4\rangle$ is conceptually either 2 or 4. The input is conceptually either $\langle 2, 4\rangle$ or $\langle 1, 2, 3\rangle$; that is, it conceptually represents 1, 2, 3, or 4. This is of course what the output is at the conceptual level.

The operator $or\_\rho_2{}^{s,t} : s\times\langle t\rangle \to \langle s\times t\rangle$ takes in a pair of type $s\times\langle t\rangle$ and pairs the first component with every item in the second component, which is an or-set. For example, $or\_\rho_2\,(1, \langle 2, 3\rangle)$ yields the or-set $\langle(1, 2), (1, 3)\rangle$. Here the input stands conceptually for a pair whose first component is 1 and whose second component is either 2 or 3. That is, the input is conceptually either

$(1,2)$ or $(1,3)$. Hence or_ρ_2 also has the important property of preserving the meaning at the conceptual level. We also use ${or\_\rho_1}^{s,t} : \langle s \rangle \times t \to \langle s \times t \rangle$ for the operator that does pairing the other way round.

The operator $\alpha^s : \{\langle s \rangle\} \to \langle \{s\} \rangle$ takes in an ordinary set containing or-sets of type s and produces an or-set containing sets of type s obtained by combining the or-sets componentwise in all possible ways. For example, $\alpha\ \{\langle 2,3 \rangle, \langle 4,5,3 \rangle\}$ produces the or-set $\langle \{2,4\}, \{2,5\}, \{2,3\}, \{3,4\}, \{3,5\}, \{3\} \rangle$. This is also an operator that preserves conceptual meaning. In the above example, the input is conceptually a set of two elements such that one of them is either 2 or 3 and the other is either 4, or 5, or 3. This is precisely what the output is conceptually. Note that sets such as $\{2\}$, $\{4\}$, etc. are not part of the output, even though $\{3\}$ is because it arises by letting both the first and second elements be 3.

As a further example, consider the result of applying α to $\{\langle 1,2 \rangle, \langle\rangle, \langle 3 \rangle\}$. It is *not* $\langle \{1,3\}, \{2,3\} \rangle$. The *correct* output is the empty or-set $\langle\rangle$. To see this, let us find out what the input is at the conceptual level. It represents a set of three elements, that are conceptually the values represented respectively by $\langle 1,2 \rangle$, $\langle\rangle$, and $\langle 3 \rangle$. Hence the first element is either 1 or 2 and the third is 3. But what is the second element? Recall that an or-set represents at the conceptual level an object that is equal to one of its elements. Since $\langle\rangle$ has no element, it does not represent any object at the conceptual level. Consequently, our input represents at the conceptual level "a set having an element which is not anything." As there is no such set, the input does not represent any object either. This coincides precisely with the meaning of the output. An item which does not represent any object at the conceptual level indicates a conceptual *inconsistency.* (But note that it is still structurally meaningful.)

The above operators provide an idea of what to include in a *structural* query language. But what kind of operators should be provided in a *conceptual* query language? Should there be an operator for testing whether two objects are conceptually equivalent? Should there be an operator for testing whether one object is among the objects denoted by a second object?

Fortunately, it is not necessary to make such chaotic "enhancements." It is found that any two objects which are conceptually equivalent can be reduced to the same object by repeated applications of the above operators. The normal form induced happens to be *independent* of the precise sequence of applications of these operators. Moreover, given the type of any object, the type of its normal form can be read off. Therefore, one can take the conceptual meaning of any object to be its normal form under the rewriting induced by the above operators. Consequently, a *conceptual query language* can be built by extending a structural query language with a single operator *normalize* which takes the input object to its normal form. A query at the conceptual level is then simply a query performed on normal forms.

Related work. Imielinski, Naqvi, and Vadaparty stressed applications of or-sets in design and planning areas and informally explained the distinction between structural and conceptual queries [17, 18]. The semantics and query language proposed by [17] are rather involved. They defined a concept of order-independence which is related to the notion of normalization but is based on assigning object identifiers, and gave conditions for order-independence. In addition, they were able to demonstrate a $co\mathcal{NP}$-complexity result for that particular proposal. In [18] they studied some intrinsic lower bounds on complexity of $\mathcal{LDL}$-style [29] queries on or-sets. The language can express queries of hyper-exponential complexity. Nevertheless, they were successful in identifying certain restricted tractable fragments that are useful in real-life applications.

Rounds [32] studied complex object databases from the situation-theoretic point of view. Connections with natural language problems motivated him to introduce the notions of conjunctive and disjunctive information which correspond exactly to our notions of sets and or-sets. He studied order relations on complex objects and their logical representations.

Organization. A query language *or-*$\mathcal{NRA}$ that cleanly integrates or-sets and more traditional types of data at the structural level is proposed in Section 2.

In Section 3 we give two semantic representations which are in the spirit of Rounds' work [32] but use simpler machinery. For example, using our representations we were able to provide a simple proof that α^s is the isomorphism of semantic domains of types $\{\langle s\rangle\}$ and $\langle\{s\}\rangle$.

A query at the conceptual level is a query on an object that is in a certain normal form. In Section 4, the normal form is properly characterized. Moreover, we show that the process of normalization is *coherent*. That is, the normal form of any object is independent of how the object is normalized. This allows us to define a query language *or-*$\mathcal{NRA}^+$ at the conceptual level by adding a new operator *normalize* to *or-*$\mathcal{NRA}$.

Since differently represented objects may have the same value (if they have the same normal form), it is clear that certain structural information is lost by normalization. In Section 5, a *losslessness* theorem is proved. Consequently, loss of structural information has no effect with respect to a large class of queries.

Conceptual queries are performed on normalized data. In Section 6, we study a few important costs of normalization. In particular, an upper bound on the number of elements in normal forms of complex objects and an upper bound on the actual size of normal forms of complex objects are given. Also significant is that we have been able to demonstrate that every definable query in *or-*$\mathcal{NRA}^+$ is at most exponential in the size of input, in contrast to the proposal of Imielinski, Naqvi, and Vadaparty [18] which contains some hyperexponential queries.

In the last section we briefly describe an implementation of the proposed languages and outline some problems for further research.

The extended abstract of this paper appeared in [24].

2 Structural Query Language

A nested relational language based on structural recursion [4, 3] and monads [27, 33] was proposed in [5]. This language is of polynomial time complexity and smoothly generalizes many approaches [2] to nested relational algebras. It is extensible and has an appealing syntax. For example, $\langle x \mid x \in normalize(DB),\ is\_cheap(x)\rangle$ selects cheap completed designs assuming that *is_cheap* and *normalize* are defined. (In Section 4, *normalize* is added as a primitive to obtain the conceptual query language.)

The algebraic version of the language is used in this paper. We denote this language by $\mathcal{NRA}(\Sigma)$ where Σ are some additional primitives like operations on integers. As observed by Wadler [33], the same syntax can be used for many "collection" types besides sets. In particular, by replacing the set operators of $\mathcal{NRA}$ by the corresponding operators for or-sets, a language for programming with or-sets can be obtained. This language is denoted by $\mathcal{NRA}_{or}$.

For example, the above query becomes $or\_\mu \circ or\_map(cond(is\_cheap, or\_\eta, K\langle\rangle \circ !)) \circ normalize$. Here *cond* is a primitive: $cond(p, t, f)(x) = t(x)$ if $p(x)$ is true and $f(x)$ otherwise. Then $cond(\ is\_cheap, or\_\eta, K\langle\rangle \circ !)\ (x)$ is $\langle x\rangle$ if x is cheap and $\langle\rangle$ otherwise. *or_map* applies it to every element in the normalized database, returning $\langle x\rangle$ for each cheap x and $\langle\rangle$ for each expensive one. or_μ flattens this or-set of or-sets, producing an or-set containing precisely the cheap completed designs.

In this section, the language for sets $\mathcal{NRA}$ and the language for or-sets $\mathcal{NRA}_{or}$ are integrated into a single language we called *the structural query language*, denoted by $or\text{-}\mathcal{NRA}$. $or\text{-}\mathcal{NRA}$ supports *structural* manipulations of complex objects containing a mixture of freely combined tuples, sets, and or-sets. This language is obtained by the union of $\mathcal{NRA}$ and $\mathcal{NRA}_{or}$ and an operator α prescribing the interaction between sets and or-sets.

Types. A type of $or\text{-}\mathcal{NRA}$ is either an object type or a function type $s \to t$, where s and t are both object types. The object types are given by the grammar: $t ::= b \mid t \times t \mid \{t\} \mid \langle t\rangle$, where b ranges over base types such as booleans and integers. Included in b is a special base type *unit* containing precisely one element. In this paper $\langle t\rangle$ stands for the or-set of type t, while $\{t\}$ is the ordinary set of type t.

Morphisms (expressions). The "morphisms" (or expressions) of $or\text{-}\mathcal{NRA}$ are formed according to the rules in Figure 1. The language is parameterized by a collection of primitives p of function type $Type(p)$. Among them are the equality tests $=_s$: $s \times s \to bool$ for each object type s, and a collection of constants c of base type $Type(c)$. Type superscripts are usually omitted because the most general type of any given morphism can be inferred; see [13].

Semantics. π_1 and π_2 are the first and second projections. ! maps everything to the unique element of type *unit*. (f, g) is pair formation, $f \circ g$ is the composition of f and g. *id* is the identity function. or_ρ_2, or_μ, and α have already been described. or_η is the singleton

Operators shared by $\mathcal{NRA}$ and $\mathcal{NRA}_{or}$

$$\frac{g : u \to s \quad f : s \to t}{f \circ g : u \to t} \qquad \frac{}{\pi_1^{s,t} : s \times t \to s} \qquad \frac{}{\pi_2^{s,t} : s \times t \to t} \qquad \frac{f : u \to s \quad g : u \to t}{(f, g) : u \to s \times t}$$

$$\frac{}{!^t : t \to unit} \qquad \frac{}{Kc : unit \to Type(c)} \qquad \frac{}{p : Type(p)} \qquad \frac{}{id^t : t \to t}$$

Operators from $\mathcal{NRA}$

$$\frac{}{\rho_2^{s,t} : s \times \{t\} \to \{s \times t\}} \qquad \frac{}{\eta^t : t \to \{t\}} \qquad \frac{}{\cup^t : \{t\} \times \{t\} \to \{t\}}$$

$$\frac{}{\mu^t : \{\{t\}\} \to \{t\}} \qquad \frac{}{K\{\}^t : unit \to \{t\}} \qquad \frac{f : s \to t}{map\ f : \{s\} \to \{t\}}$$

Operators from $\mathcal{NRA}_{or}$

$$\frac{}{or\_\rho_2{}^{s,t} : s \times \langle t \rangle \to \langle s \times t \rangle} \qquad \frac{}{or\_\eta^t : t \to \langle t \rangle} \qquad \frac{}{or\_\cup^t : \langle t \rangle \times \langle t \rangle \to \langle t \rangle}$$

$$\frac{}{or\_\mu^t : \langle\langle t \rangle\rangle \to \langle t \rangle} \qquad \frac{}{K\langle\rangle^t : unit \to \langle t \rangle} \qquad \frac{f : s \to t}{or\_map\ f : \langle s \rangle \to \langle t \rangle}$$

Interaction of sets and or-sets

$$\frac{}{\alpha^t : \{\langle t \rangle\} \to \langle\{t\}\rangle}$$

Figure 1: Syntax of $or\text{-}\mathcal{NRA}$

formation: $or\_\eta(x) = \langle x \rangle$. $or\_\cup$ makes the union of two or-sets. $or\_map(f)$ applies f to all elements of an or-set. $K\langle\rangle$ produces an empty or-set. or_ρ_1 has been omitted because it is definable as $or\_map(\pi_2, \pi_1) \circ or\_\rho_2 \circ (\pi_2, \pi_1)$. The operators from $\mathcal{NRA}$ have similar meaning for the usual sets.

We have included $K\langle\rangle$, the morphism which produces the empty or-set, in *or*-$\mathcal{NRA}$. We note that if f is a morphism of *or*-$\mathcal{NRA}$ such that $K\langle\rangle$ does not occur in it and such that each primitive p in it does not produce the empty or-set, then f applied to any complex object x not containing any empty or-set yields a complex object $f(x)$ containing no empty or-set.

The primitive α is essentially a translation of conjunctive normal form into disjunctive normal form. This operation may be very expensive. Indeed, if its argument is a collection of n two-element or-sets, all $2n$ elements being distinct, then α produces an or-set containing 2^n n-element sets. Several query languages have expensive exponential-cost operations. For example, in the Abiteboul-Beeri algebra [1, 5], one of the primitives is $powerset$: $\{t\} \to \{\{t\}\}$ which takes a set and returns the set of all its subsets. The result that we are going to formulate can be intuitively understood as follows: the expressive power of α is that of $powerset$. However, $powerset$ does not use the $\langle\rangle$ type constructor. To be able to speak of the equivalence of expressive power of languages one of which uses or-sets and the other does not, for technical purposes only, we introduce the functions $or\_to\_set : \langle t \rangle \to \{t\}$ and $set\_to\_or : \{t\} \to \langle t \rangle$ with the obvious semantics: $or\_to\_set(\langle x_1, \ldots, x_n \rangle) = \{x_1, \ldots, x_n\}$ and $set\_to\_or(\{x_1, \ldots, x_n\}) = \langle x_1, \ldots, x_n \rangle$. We remark here that, if or_to_set and set_to_or are given, then $\mathcal{NRA}$ and $\mathcal{NRA}_{or}$ are interdefinable. That is, $\mathcal{NRA}(or\_to\_set, set\_to\_or) \cong \mathcal{NRA}_{or}(or\_to\_set, set\_to\_or)$.

Proposition 2.1 $\mathcal{NRA}(or\_to\_set, set\_to\_or, \alpha) \cong \mathcal{NRA}(or\_to\_set, set\_to\_or, powerset)$.

Proof. It can be seen that $powerset$ is definable as follows:

$$powerset = \; or\_to\_set \circ \alpha \circ map(or\_\cup \circ (or\_\eta \circ K\{\}\circ !, or\_\eta \circ \eta))$$

Conversely, we must show that α is definable in $\mathcal{NRA}(or\_to\_set, set\_to\_or, powerset)$. For the sake of clarity we use *cond* to show that α is definable. A clumsier proof that does not use *cond* is also possible. It is known [5] that the test for equal cardinality can be implemented in $\mathcal{NRA}(powerset)$. To check whether $|X| \leq |Y|$, notice that

$$\mu \circ map(\lambda Z.cond(equal\_card?(X, Z), X, \{\}))(powerset(Y))$$

returns X if $|X| \leq |Y|$ and $\{\}$ otherwise, thus giving us the test for lesser cardinality.

Now, given an input of type $\{\langle t \rangle\}$, first apply $map(or\_to\_set)$ to it and then flatten the result, thus obtaining the set of elements that occur in the input. Applying $powerset$ now gives the set of all sets of those elements. A set of elements of the input makes it to the output if and only if two conditions hold. First, its cardinality does not exceed the cardinality of the input, which is the number of or-sets. Second, it has a nonempty intersection with any element of the input, unless the input is $\{\}$. Since selection, lesser cardinality test, intersection and test for nonemptiness are definable in $\mathcal{NRA}(powerset)$ (see [5] and above), selection over the powerset followed by an application of *set_to_or* yields the desired result. $\Box$

3 Partial Information and Or-sets

In this section we address some semantic issues. The presence of or-sets in a database means the presence of partial information. We assume that partiality can be expressed by means of a partial order on database objects. That is, $x \leq y$ expresses the fact that x is more partial than y or y is more informative than x. The idea of using partially ordered sets to model partial information has been around since early 80s: Codd's tables, for example, can be captured by so-called flat domains which are obtained from unordered sets by adding a unique bottom element (null). An approach of having three kinds of nulls — unknown, nonexistent, existent unknown — is another example of ordering on objects. In fact, a general approach to the treatment of partial information as ordering on the set of objects was proposed in [7] and further developed in [6, 19, 21]. We remark here that this approach is also suitable for databases *without* partial information. In such a case, values of base types are totally unordered.

Assume that orders on values of base types are given. It is clear how to order pairs: $(x, y) \leq (x', y')$ iff $x \leq x'$ and $y \leq y'$. However, there is no immediate answer to the question of how to extend the ordering to set and or-set types. In [7, 6, 19, 32] two ways to extend an ordering to subsets of a partially ordered set were studied. Let $\langle X, \leq \rangle$ be a poset and $A, B \subseteq X$. *The Hoare* ($\leq^\flat$) and *the Smyth* ($\leq^\sharp$) orderings are defined as follows:

$$A \leq^\flat B \Leftrightarrow \forall a \in A\ \exists b \in B:\ a \leq b$$

$$A \leq^\sharp B \Leftrightarrow (\forall b \in B\ \exists a \in A:\ a \leq b)\ \&\ (B = \emptyset \Rightarrow A = \emptyset)$$

Traditionally the condition $B = \emptyset \Rightarrow A = \emptyset$ is omitted because the Smyth powerdomain does not contain the empty set. Observe that if X is totally unordered, $\leq^\flat$ is the subset and $\leq^\sharp$ is the superset ordering on non-empty sets. The Hoare ordering was also used in [15] to order relations with partial information. We will try to justify using $\leq^\flat$ to order values of set types and $\leq^\sharp$ to order values of or-set types.

Assume that a set $A \subseteq X$ is given. How can we improve our knowledge about the real world situation represented by this set? There are two ways to do so. First, by replacing an element $a \in A$ by a set A' of elements greater than a. For example, if a record [Name $\Rightarrow$ $\perp$, Office $\Rightarrow$ '515'] is contained in the database, we can improve our knowledge about the office assignment by replacing this record by [Name $\Rightarrow$ 'Joe', Office $\Rightarrow$ '515'] and [Name $\Rightarrow$ 'Mary', Office $\Rightarrow$ '515']. Second, we can add an element to the set. For example, adding a record [Name $\Rightarrow$ 'Bill', Office $\Rightarrow$ '212'] gives us more information about office allocation.

Define a binary relation $\rightsquigarrow$ on subsets of X as follows: $A \rightsquigarrow (A - \{a\}) \cup A'$, where $a \leq a'$ for all $a' \in A'$, and $A \rightsquigarrow A \cup \{a\}$. A set B is said to be *more informative* than A, denoted $A \overset{*}{\rightsquigarrow} B$, if B can be obtained from A by a sequence of transformations $\rightsquigarrow$. In other words, $\overset{*}{\rightsquigarrow}$ is the reflexive-transitive closure of $\rightsquigarrow$.

Similarly for or-sets we define $\mapsto$ by $A \mapsto (A - \{a\}) \cup A'$, where $a \leq a'$ for all $a' \in A'$, and $A \mapsto A - \{a\}$ provided that $A - \{a\}$ is not empty (removing an element from an or-set makes it more informative). Again, $\overset{*}{\mapsto}$ is defined as the reflexive-transitive closure of $\mapsto$.

Proposition 3.1 $\overset{*}{\rightsquigarrow}$ *coincides with* $\leq^\flat$ *and* $\overset{*}{\mapsto}$ *coincides with* $\leq^\sharp$.

Proof. First notice that $\rightsquigarrow\ \subseteq \leq^\flat$ and $\mapsto \subseteq \leq^\sharp$. Therefore, transitivity of $\leq^\flat$ and $\leq^\sharp$ implies $\overset{*}{\rightsquigarrow} \subseteq \leq^\flat$ and $\overset{*}{\mapsto} \subseteq \leq^\sharp$.

To prove the reverse inclusion, let $A \leq^\flat B$. The case of empty sets is obvious, so assume $A, B \neq \emptyset$. Let $B_a = \{b \in B \mid a \leq b\}$ and $B_A = \bigcup_{a \in A} B_a$. Notice that $B_A \neq \emptyset$. For each $a \in A$, apply the following transformations to A: $A \rightsquigarrow (A - \{a\}) \cup (B_a \cup \{a\})$ for each $a \in A$ in any order. This shows $A \overset{*}{\rightsquigarrow} (A \cup B_A)$. For any $a \in A$, pick $b_a \in B_a$ and apply transformations $A \cup B_A \rightsquigarrow ((A \cup B_A) - \{a\}) \cup \{b_a\}$ in any order, thus obtaining $A \overset{*}{\rightsquigarrow} B_A$. Finally, if $B - B_A \neq \emptyset$ and $B - B_A = \{b_1, \ldots, b_k\}$, $B_A \rightsquigarrow B_A \cup \{b_1\} \rightsquigarrow \ldots \rightsquigarrow B_A \cup \{b_1, \ldots, b_k\} = B$, i.e. $A \overset{*}{\rightsquigarrow} B$. This shows $\leq^\flat \subseteq \overset{*}{\rightsquigarrow}$. The proof of $\leq^\sharp \subseteq \overset{*}{\mapsto}$ is similar. $\square$

This proposition justifies the semantics of types defined below. Notice that the semantics for or-sets is given in such a way that the empty or-set is incomparable with any other or-set. This matches the intention that the empty or-set represents inconsistency.

- For each base type b a poset $\langle [\![b]\!], \leq_b \rangle$ is given;
- $[\![s \times t]\!] = \langle [\![s]\!] \times [\![t]\!], \leq_s \times \leq_t \rangle$;
- $[\![\{t\}]\!] = \langle \mathbf{P}_{\text{fin}}([\![t]\!]), \leq_t^\flat \rangle$;
- $[\![\langle t \rangle]\!] = \langle \mathbf{P}_{\text{fin}}([\![t]\!]), \leq_t^\sharp \rangle$.

In several papers dealing with partial information in databases it was proposed that instances of type $\{t\}$ be restricted to those containing no comparable elements, commonly called *antichains*, see [7, 19]. For example, if one field of a record plays the role of the object identifier (oid), then instead of having two comparable elements with the same oid their join should be taken, provided the records with the same oid are consistent. One way to obtain an antichain from an arbitrary finite set is to take all its maximal elements. Dually, we can take its minimal elements. Antichains thus obtained will be denoted by $\max_\leq A$ and $\min_\leq A$ or just $\max A$ and $\min A$ if the ordering is understood. We suggest using max for the usual sets and min for or-sets

[21, 32]. Then the relations $\rightsquigarrow$ and $\mapsto$ must be redefined as follows: $A \rightsquigarrow_a \max((A-a)\cup A')$, $A \rightsquigarrow_a \max(A\cup a)$ and $A \mapsto_a \min((A-a)\cup A')$, $A \mapsto_a \min(A-a)$. As before, we define $\overset{*}{\rightsquigarrow}_a$ and $\overset{*}{\mapsto}_a$ to be the transitive closure of $\rightsquigarrow_a$ and $\mapsto_a$ respectively.

Proposition 3.2 *On the family of finite antichains of $\langle X, \leq\rangle$, $\overset{*}{\rightsquigarrow}_a$ coincides with $\leq^\flat$ and $\overset{*}{\mapsto}_a$ coincides with $\leq^\sharp$.*

Proof. Again, as in the proof of Proposition 3.1, only the case of nonempty sets should be considered and only one direction, namely $\leq^\flat \subseteq \overset{*}{\rightsquigarrow}_a$ and $\leq^\sharp \subseteq \overset{*}{\mapsto}_a$ must be proved as the other direction is immediate. We also need the following ordering on sets, called *the Plotkin ordering* (cf. [11]): $A \leq^\natural B \Leftrightarrow A \leq^\flat B$ and $A \leq^\sharp B$.

Let $A, B \neq \emptyset$, $A\cap B = \emptyset$. Define B_A as in the proof of Proposition 3.1. Similarly, $A_b = \{a \in A \mid a \leq b\}$ and $A_B = \bigcup_{b\in B} A_b$.

Claim 1. Let $A \leq^\natural B$, $A\cap B=\emptyset$. Then $A \overset{*}{\rightsquigarrow}_a B$ and, moreover, only elements of $A\cup B$ are used in the transformations.

Proof of Claim 1. We proceed by induction on the size of B. The base case $|B|= 1$ is obvious. If $|B|> 1$, let B' be a minimal subset of B such that $A \leq^\natural B'$. Our goal is to show that there exists $b \in B'$ such that $A - A_b \leq^\natural B' - b$. Then, by the induction hypothesis, $A - A_b \overset{*}{\rightsquigarrow}_a B' - \{b\}$. Since the left hand side and the right hand side remain antichains if one adds any subset of A_b to them, we obtain $A \overset{*}{\rightsquigarrow}_a A_b \cup (B' - \{b\})$ (the same transformations can be used). Since $A_b \cup (B' - \{b\}) \overset{*}{\rightsquigarrow}_a B'$, $A \overset{*}{\rightsquigarrow}_a B'$ follows, and adding elements of $B - B'$ gives us $A \overset{*}{\rightsquigarrow}_a B$.

Assume that there is no $b \in B'$ such that $A - A_b \leq^\natural B' - b$. Since $A - A_b \leq^\flat B' - b$ holds, this means $A - A_b \not\leq^\sharp B' - b$ for any $b \in B$. Fix an element $b \in B$. Since $B' - b \neq \emptyset$, we can find an element $b' \in B' - b$ such that $a \not\leq b'$ for all $a \in A - A_b$. In other words, $A_{b'} \subseteq A_b$. We claim that $A \leq^\natural B' - b'$. Indeed, $A \leq^\sharp B' \leq^\sharp B' - b'$. Now consider $a \in A$. We must show that there is $b_0 \in B' - b'$ such that $a \leq b_0$. Since $A \leq^\flat B'$, there is $b_0 \in B'$ such that $a \leq b_0$. If $b_0 \neq b'$, we are done. If $b_0 = b'$, then $a \in A_{b'} \subseteq A_b$ and hence $a \leq b \in B' - b'$. Thus, $A \leq^\flat B' - b'$ and hence $A \leq^\natural B' - b'$, which contradicts the minimality of B'. This contradiction finishes the proof of Claim 1.

Claim 2. Let $A \leq^\natural B$, $A\cap B = \emptyset$. Then $A \overset{*}{\mapsto}_a B$ and, moreover, only elements of $A\cup B$ are used in the transformations.

Proof of Claim 2. The proof is similar to that of Claim 1. We use induction on $|B|$. Since removal is now allowed, assume w.l.o.g. that no proper subset of A is less than B w.r.t. $\leq^\natural$. We claim that there exists $a \in A$ such that $A - \{a\} \leq^\natural B - B_a$. Suppose not; then for every $a \in A$ there exists $a_1 \in A$ such that $B_{a_1} \subseteq B_a$. Continuing, we obtain $B_a \supseteq B_{a_1} \supseteq B_{a_2} \supseteq \ldots$. Since all the sets are finite, $B_{a_i} = B_{a_j}$ for some distinct a_i and a_j which contradicts minimality of A for $A - \{a_i\} \leq^\natural B$. Now, given $a \in A$ such that $A - \{a\} \leq^\natural B - B_a$, apply the hypothesis to $A - \{a\}$

and $B - B_a$ and observe that a is not under any element of $B - B_a$. Hence, $A \stackrel{*}{\mapsto}_a (B - B_a) \cup \{a\}$ since only elements of $(A - \{a\}) \cup (B - B_a)$ were used in transformation. $(B - B_a) \cup \{a\} \stackrel{*}{\mapsto}_a B$ finishes the proof of Claim 2.

Claim 3. Let $A \stackrel{*}{\leadsto}_a B$ (or $A \stackrel{*}{\mapsto}_a B$) and all $\leadsto$ and $\mapsto$ transformations use only elements of A and B. If C is a finite set such that both $A \cup C$ and $B \cup C$ are antichains, then $A \cup C \stackrel{*}{\leadsto}_a B \cup C$ (or $A \cup C \stackrel{*}{\mapsto}_a B \cup C$).

Proof of Claim 3. Clearly, C does not interact with any $\stackrel{*}{\leadsto}_a$ or $\stackrel{*}{\mapsto}_a$ transformation, provided they use only elements of $A \cup B$.

Now, let $A \leq^\flat B$. Since A and B are antichains, for $A' = A - B$ and $B' = B - A$ one has $A' \leq^\flat B'$. Therefore, $A' \leq^\natural B'_A$ and by Claim 1 $A' \stackrel{*}{\leadsto}_a B'_A$. Moreover, all transformations use only elements from $A' \cup B'_A$. Then, by claim 3, $A \stackrel{*}{\leadsto}_a B'_A \cup (A \cap B)$. Adding elements to the right hand side one obtains $A \stackrel{*}{\leadsto}_a B$. The proof that $A \leq^\sharp B$ implies $A \stackrel{*}{\mapsto}_a B$ is similar and it relies on Claims 2 and 3. The proposition is proved. $\Box$

This proposition shows that if we deal with antichains, we can change the last two clauses in the inductive definition of the semantics of types to

- $[\![\{t\}]\!]_a = \langle \mathbf{A}_{\mathrm{fin}}([\![t]\!]), \leq^\flat_t \rangle$;
- $[\![\langle t \rangle]\!]_a = \langle \mathbf{A}_{\mathrm{fin}}([\![t]\!]), \leq^\sharp_t \rangle$;

where $\mathbf{A}_{\mathrm{fin}}(X)$ is the set of finite antichains of X. It is clear how to define the semantics of *or-$\mathcal{NRA}$* expressions if either semantics for types is used. In the case of the antichain semantics, if an application produces a set (or or-set), max (or min) operation is used to make the resulting object into an antichain.

The operator α in the case of the antichain semantics requires some care: $\alpha_a \equiv [\![\alpha]\!]_a$ is a function from $[\![\{\langle t \rangle\}]\!]_a$ to $[\![\langle \{t\} \rangle]\!]_a$. Given an element of $[\![\{\langle t \rangle\}]\!]_a$, i.e. an antichain $\mathcal{A} = \{A_1, \ldots, A_n\}$ w.r.t. $\leq^\flat$, of antichains from $[\![t]\!]_a$, let $A_i = \{a^i_1, \ldots, a^i_{n_i}\}$. Let $\mathcal{F}_\mathcal{A}$ be the set of all choice functions $f : \{1, \ldots, n\} \to \mathbb{N}$ such that $1 \leq f(i) \leq n_i$. For $f \in \mathcal{F}_\mathcal{A}$, $f(\mathcal{A})$ is defined to be $\{a^1_{f(1)}, \ldots, a^n_{f(n)}\}$. Then

$$\alpha_a(\mathcal{A}) = \min_{f \in \mathcal{F}_\mathcal{A}} {}_{\leq^\flat}(\max f(\mathcal{A}))$$

Furthermore, the result that iterated powerdomains are isomorphic [9, 14] can now be given a very simple description along the lines of [20]:

Theorem 3.3 *α_a establishes an isomorphism between $[\![\{\langle t \rangle\}]\!]_a$ and $[\![\langle \{t\} \rangle]\!]_a$. The converse β_a is*

$$\beta_a(\mathcal{A}) = \max_{f \in \mathcal{F}_\mathcal{A}} {}_{\leq^\sharp}(\min f(\mathcal{A})), \quad \mathcal{A} \in [\![\langle \{t\} \rangle]\!]_a.$$

Proof. We have to show that α_a maps $[\![\{\langle t\rangle\}]\!]_a$ to $[\![\langle\{t\}\rangle]\!]_a$, β_a maps $[\![\langle\{t\}\rangle]\!]_a$ to $[\![\{\langle t\rangle\}]\!]_a$ and α_a and β_a are mutually inverse and monotone. The first two claims follow immediately from the definitions of α_a and β_a. To complete the proof, show that α_a is monotone and $\beta_a \circ \alpha_a = \mathsf{id}$. By duality the proof of monotonicity of β_a and $\alpha_a \circ \beta_a = \mathsf{id}$ can be obtained.

We start with two easy observations. If Y_1 and Y_2 are finite subsets of an arbitrary poset, then (1) $Y_1 \leq^\flat Y_2$ iff $\max Y_1 \leq^\flat \max Y_2$ and (2) $Y_1 \leq^\sharp Y_2$ iff $\min Y_1 \leq^\sharp \min Y_2$.

Throughout this proof, $\mathcal{A}$ is defined as above, i.e. $\mathcal{A} = \{A_1, \dots, A_n\}$ and each A_i consists of elements a^i_j, $j = 1, \dots, k_i$.

Claim 1. α_a is monotone.

Proof of Claim 1. Let $\mathcal{A}, \mathcal{B} = \{B_1, ..., B_m\} \in [\![\{\langle t\rangle\}]\!]_a$ and $\mathcal{A} \leq^\flat \mathcal{B}$. We must prove that $\alpha_a(\mathcal{A}) \leq^\sharp \alpha_a(\mathcal{B})$. In view of the two observations above, it is enough to show that for any $f \in \mathcal{F}_\mathcal{B}$ there exists $g \in \mathcal{F}_\mathcal{A}$ such that $g(\mathcal{A}) \leq^\flat f(\mathcal{B})$. Since for each $i = 1, \dots, n$ there exists j_i such that $A_i \leq^\sharp B_{j_i}$, there is an element $a^i_{p_i} \in A_i$ such that $a^i_{p_i} \leq b^{j_i}_{f(j_i)}$. Let $g(i) = p_i$. Then for this function g one has $\{a^i_{g(i)} \mid i = 1, \dots, n\} \leq^\flat \{b^i_{f(i)} \mid i = 1, \dots, m\}$, i.e. $g(\mathcal{A}) \leq^\flat f(\mathcal{B})$. Claim 1 is proved.

Let $\mathcal{A} \in [\![\{\langle t\rangle\}]\!]_a$ and $\mathcal{B} = \{B_1, \dots, B_m\} = \alpha_a(\mathcal{A}) \in [\![\langle\{t\}\rangle]\!]_a$. By the two observations above, to show that $\beta_a \circ \alpha_a = \mathsf{id}$, i.e. that $\beta_a(\mathcal{B}) = \mathcal{A}$, it suffices to prove two claims.

Claim 2. For any $f \in \mathcal{F}_\mathcal{B}$ there exists $A_i \in \mathcal{A}$ such that $f(\mathcal{B}) \leq^\sharp A_i$.

Claim 3. Every A_i is in $\beta_a(\mathcal{B})$.

Proof of Claim 2. Let $\mathcal{C}$ be the collection of all sets $f(\mathcal{A})$ where $f \in \mathcal{F}_\mathcal{A}$; $\mathcal{C} = \{C_1, \dots, C_k\}$. Then for any $g \in F_\mathcal{C}$, there exists $A_i \in \mathcal{A}$ such that A_i is contained in $g(\mathcal{C})$ because, if this is not the case, for any $A_i \in \mathcal{A}$ there exists $j_i \leq k_i$ such that $a^i_{j_i} \in A_i$ and, for any $f \in \mathcal{F}_\mathcal{A}$, g on $f(\mathcal{A})$ picks an element different from $a^i_{j_i}$. If we define f_0 such that $f_0(i) = j_i$, g may pick only elements of form $a^i_{j_i}$ on $f_0(\mathcal{A})$, a contradiction. Therefore, $g(\mathcal{C}) \leq^\sharp A_i$ for some i.

Let $f \in \mathcal{F}_\mathcal{B}$. Let H be the set of functions in $\mathcal{F}_\mathcal{A}$ that correspond to the elements of $\mathcal{B} = \alpha_a(\mathcal{A})$ or, in other words, $\max h(\mathcal{A}) \in \mathcal{B}$ for $h \in H$. Then, for any $h' \in \mathcal{F}_\mathcal{A} - H$, there exists a function $h \in H$ such that $\max h(\mathcal{A}) \leq^\flat \max h'(\mathcal{A})$, i.e. $h(\mathcal{A}) \leq^\flat h'(\mathcal{A})$. Since $h \in H$, $\max h(\mathcal{A}) \in \mathcal{B}$, i.e. $\max h(\mathcal{A}) = B_i$. If $f(i) = j$, then there is an element in $h'(\mathcal{A})$ that is greater than b^i_j. Define a function $g \in F_\mathcal{C}$ to coincide with f on those C_i's that are given by the functions in H. On C_i that corresponds to $f \in \mathcal{F}_\mathcal{A} - H$, let g pick an element which is greater than some b^i_j where $f(i) = j$ (we have just shown it can be done). Then $f(\mathcal{B}) \leq^\sharp \{c^i_{g(i)} \mid i = 1, \dots, k\} = g(\mathcal{C})$. We know that there exists $A_i \in \mathcal{A}$ such that $g(\mathcal{C}) \leq^\sharp A_i$. Thus, $f(\mathcal{B}) \leq^\sharp A_i$. Claim 2 is proved.

Proof of Claim 3. We prove that for any $a^i_j \in A_i$ there exists $B_l \in \mathcal{B}$ such that $a^i_j \in B_l$. Consider the set $F^{ij}_\mathcal{A}$ of functions $f \in \mathcal{F}_\mathcal{A}$ such that $f(i) = j$. If for no $f \in F^{ij}_\mathcal{A}$: $a^i_j \in \max f(\mathcal{A})$, then there exists $A_p \in \mathcal{A}$ such that all elements of A_p are greater than a^i_j, i.e. $A_i \leq^\sharp A_p$. This contradicts our assumption that $\mathcal{A}$ is an antichain w.r.t. $\leq^\sharp$. Hence, $a^i_j \in \max f(\mathcal{A})$ for at least one function in $F^{ij}_{\mathcal{A}}$. Since $\mathcal{A}$ is an antichain, for any $p \neq i$ there exists $a^p_q \in A_p$ which is not greater than any element of A_i. Change f to pick such an element for any $p \neq i$. Then a^i_j is still in $\max f(\mathcal{A})$. There exists a function $f' \in \mathcal{F}_{\mathcal{A}}$ such that $\max f'(\mathcal{A}) \leq^\flat \max f(\mathcal{A})$ and $\max f'(\mathcal{A}) \in \alpha_a(\mathcal{A})$. If $f'(i) = j' \neq j$, then, since $f'(\mathcal{A}) \leq^\flat f(\mathcal{A})$ and A_i is an antichain, $a^i_{j'} \leq a^p_q$ for some p and q, where $p \neq i$. But this contradicts the definition of f. Hence, $f'(i) = j$ and $a^i_j \in \max f'(\mathcal{A})$ because $a^i_j \in \max f(\mathcal{A})$. Since $\max f'(\mathcal{A}) = B_l$ for some index l, $a^i_j \in B_l \in \mathcal{B}$.

Let $\mathcal{B}'$ be the collection of elements of $\mathcal{B}$ that contain elements of A_i. Then we can define a function $f \in \mathcal{F}_{\mathcal{B}}$ on elements of $\mathcal{B}'$ to pick all elements of A_i. Each $B_j \in \mathcal{B} - \mathcal{B}'$ either contains an element of A_i or contains an element which is greater than some $a^i_p \in A_i$. Let f pick any such element. Then $\min f(\mathcal{B}) = A_i$. Suppose $A_i \notin \beta_a(\mathcal{B})$. Then $A_i \leq^\sharp \min g(\mathcal{B})$ for some function $g \in \mathcal{F}_{\mathcal{B}}$ such that $\min g(\mathcal{B}) \in \beta_a(\mathcal{B})$. By Claim 2, $g(\mathcal{B}) \leq^\sharp A_j$ for some A_j. Hence, $\min g(\mathcal{B}) \leq^\sharp A_j$ and since $\mathcal{A}$ is an antichain w.r.t. $\leq^\sharp$, $A_i = A_j = \min g(\mathcal{B}) \in \beta_a(\mathcal{B})$. This finishes the proof of Claim 3 and the theorem. $\square$

It was shown in [34] that the orderings $\leq^\flat$ and $\leq^\sharp$ can be given a logical interpretation. Motivated by applications in the semantics of concurrent programming, Winskel used the modal connectives $\Box$ and $\Diamond$ to describe $\leq^\flat$ and $\leq^\sharp$. Rounds [32] used a similar logic to show the interaction between derivable properties of complex objects and their ordering. Here we present what we believe is the simplest interpretation of the logics of [32, 34] for complex objects with or-sets.

Start with an unspecified language $\mathcal{L}$ that contains the symbol $\vee$ for disjunction but does not contain $\&, \Box$ and $\Diamond$. With each element $x \in [\![b]\!]$, where b is a base type, associate a collection of formulae in $\mathcal{L}$ closed under $\vee$, called the *theory of* x and denoted by $\mathrm{Th}(x)$, in such a way that $x < y$ implies $\mathrm{Th}(x) \supset \mathrm{Th}(y)$ and $x \neq y$ implies $\mathrm{Th}(x) \neq \mathrm{Th}(y)$. For example, if $[\![b]\!]$ is a *flat domain*, i.e. an unordered collection of values with added bottom element $\perp$ which is less than anything else, the above requirement says that theories of distinct nonbottom elements do not coincide and the theory of $\perp$ contains all other theories (i.e. bottom implies everything).

The theory of a pair is a collection of pairs of statements from the theories of the components. The theory of a set is informally defined as those facts that are true of all elements of the set. A theory of an or-set contains facts that are true of at least one element of the or-set. These descriptions are the unary connectives in modal logic usually denoted by $\Box$ and $\Diamond$.

Now we can give a formal definition of theories of objects in an extended language $\mathcal{L} \cup \{\&, \Box, \Diamond\}$. A theory of an object x, $\mathrm{Th}(x)$, is the minimal collection of formulae which contains

- $\{\varphi_1 \& \varphi_2 \mid \varphi_i \in \mathrm{Th}(x_i), i = 1, 2\}$ if $x = (x_1, x_2)$;
- $\{\Box\varphi \mid \forall i : \varphi \in \mathrm{Th}(x_i)\}$ if $x = \{x_1, \ldots, x_n\}$;

- $\{\Diamond\varphi \mid \exists i : \varphi \in \mathrm{Th}(x_i)\}$ if $x = \langle x_1, \ldots, x_n \rangle$,

and, together with any $\varphi \in \mathrm{Th}(x)$, all formulae $\varphi \vee \psi$ (that is, if φ is in the theory of x, so are all the disjunctions $\varphi \vee \psi$).

Proposition 3.4 *Given two objects x, y of the same type, $x \leq y$ iff* $\mathrm{Th}(x) \supseteq \mathrm{Th}(y)$.

Proof. We proceed by induction on the type of x and y. The base case follows immediately from the definition. The case of pair is easy. Let $x = \{x_1, \ldots, x_n\}$ and $y = \{y_1, \ldots, y_m\}$. Then $x \leq y$ means $x \leq^\flat y$. If $\Box\varphi \in \mathrm{Th}(y)$, then for all $i = 1, \ldots, m$: $\varphi \in \mathrm{Th}(y_i)$. Given any x_j, there exists y_i such that $x_j \leq y_i$; hence $\varphi \in \mathrm{Th}(x_j)$ and therefore $\Box\varphi \in \mathrm{Th}(x)$. Conversely, let $\mathrm{Th}(x) \supseteq \mathrm{Th}(y)$. Suppose that $x \not\leq^\flat y$, i.e. there exists x_i such that $x_i \leq y_j$ for no y_j. Then, by the hypothesis, there exists a formula $\varphi_j \in \mathrm{Th}(y_j)$ such that $\varphi_j \notin \mathrm{Th}(x_i)$. Let $\varphi = \varphi_1 \vee \ldots \vee \varphi_m$. Then $\varphi \in \mathrm{Th}(y_j)$ for all $j = 1, \ldots, m$. Therefore, $\Box\varphi \in \mathrm{Th}(y) \subseteq \mathrm{Th}(x)$, i.e. $\varphi_1 \vee \ldots \vee \varphi_m \in \mathrm{Th}(x_i)$ which means that for at least one j: $\varphi_j \in \mathrm{Th}(x_i)$. This contradiction proves $x \leq^\flat y$. A similar proof for the case of or-sets which is based on the properties of $\leq^\sharp$ is omitted. $\Box$

Since $X \leq^\flat Y$ iff $\max X \leq^\flat \max Y$ and $X \leq^\sharp Y$ iff $\min X \leq^\sharp \min Y$, Proposition 3.4 is true if either $[\![\cdot]\!]$ or $[\![\cdot]\!]_a$ semantics is used.

4 Conceptual Query Language and Normalization

As we have pointed out, there are two levels of manipulation of objects — structural and conceptual. This section is devoted to the query language for the conceptual level.

We start with an example. Given an object $x = (\{\langle 1, 2\rangle, \langle 3\rangle\}, \langle 1, 2\rangle)$ of type $\{\langle int\rangle\} \times \langle int\rangle$. Denote the first component by y. Applying or_ρ_2 to x first yields $\langle (y, 1), (y, 2)\rangle$ which is an object of type $\langle \{\langle int\rangle\} \times int\rangle$. Applying $or\_map(\alpha \circ \pi_1, \pi_2)$ yields an object

$$\langle (\langle \{1, 3\}, \{2, 3\}\rangle, 1), (\langle \{1, 3\}, \{2, 3\}\rangle, 2)\rangle$$

of type $\langle \langle \{int\}\rangle \times int\rangle$. Finally, applying $or\_\mu \circ or\_map(or\_\rho_1)$ yields

$$\langle (\{1, 3\}, 1), (\{1, 3\}, 2), (\{2, 3\}, 1), (\{2, 3\}, 2)\rangle$$

of type $\langle \{int\} \times int\rangle$. This can be considered as a conceptual level object because all the possibilities are listed.

However, one could have used another strategy to list all the possibilities. For example, to apply $(\alpha \circ \pi_1, \pi_2)$ first to obtain an object of type $\langle \{int\}\rangle \times \langle int\rangle$ and then $or\_\mu \circ or\_map(or\_\rho_1) \circ or\_\rho_2$

to obtain an object of type $\langle \{int\} \times int \rangle$. It is easy to check that such a strategy results in precisely the same object as the previous one.

In fact, there is a general result saying that each type has a unique representation at the conceptual level such that no or-set type occurs in the type expression except as the outermost type constructor. For reasons that should emerge shortly we call such a type *a normal form*. Furthermore, for each object of type t there exists its unique representation at the conceptual level whose type is the normal form of t.

To state these results precisely, we need some definitions about rewrite systems [8]. If a signature is fixed, a *rewrite system* is a set of rules of form $\tau_1 \longrightarrow \tau_2$ where τ_1, τ_2 are terms. If σ is obtained from τ by rewriting a subterm of τ, we also write $\tau \longrightarrow \sigma$. If σ is obtained from τ by a (possibly empty) sequence of applications of rewrite rules, we write $\tau \twoheadrightarrow \sigma$.

A term τ is called a *normal form* if there is no other term σ such that $\tau \longrightarrow \sigma$. A rewrite system is called *terminating* if there is no infinite sequence of terms $\tau_1 \longrightarrow \tau_2 \longrightarrow \ldots$. It is called *Church-Rosser* if, whenever $\tau \twoheadrightarrow \tau_1$ and $\tau \twoheadrightarrow \tau_2$, there exists a term τ' such that $\tau_1 \twoheadrightarrow \tau'$ and $\tau_2 \twoheadrightarrow \tau'$. In a Church-Rosser terminating system for every term τ there exists a unique normal form $nf(\tau)$ such that $\tau \twoheadrightarrow nf(\tau)$.

Now we introduce the rewrite rules for type expressions:

$$t \times \langle s \rangle \longrightarrow \langle t \times s \rangle \qquad \langle t \rangle \times s \longrightarrow \langle t \times s \rangle$$

$$\langle \langle t \rangle \rangle \longrightarrow \langle t \rangle \qquad \{\langle t \rangle\} \longrightarrow \langle \{t\} \rangle$$

Proposition 4.1 *The above rewrite system is terminating and Church-Rosser. The normal form $nf(t)$ for type t can be found as follows. If t does not use $\langle\rangle$, then $nf(t) = t$. Otherwise, remove all angle brackets from t. If the resulting type is t', then $nf(t) = \langle t' \rangle$.*

Proof. To show that the rewrite system is terminating, define the following function on types. Considering types as their derivation trees, let k_i be the number of occurrences of $\langle\rangle$ on the ith level of the derivation tree of type t. If the height of the derivation tree is n, define $\varphi(t)$ as $\Sigma_{i=1}^{n} k_i \cdot i$. It is easy to see that if $t \longrightarrow t_0$, then $\varphi(t) > \varphi(t_0)$. Hence, any rewriting terminates.

To prove Church-Rosserness, one has to find the so-called *critical pairs* [8], which in essence are pairs of terms that can give rise to ambiguity in rewriting, and show that for any critical pair (τ_1, τ_2) there exists a term τ such that $\tau_1 \twoheadrightarrow \tau$ and $\tau_2 \twoheadrightarrow \tau$. We refer the interested reader to [8] for the definitions and proof of the critical pair lemma. A straightforward analysis of our rewrite system reveals the following critical pairs: 1) $(\langle \{\langle t \rangle\} \rangle, \{\langle t \rangle\})$; 2) $(\langle t \times \langle s \rangle \rangle, t \times \langle s \rangle)$; 3) $(\langle \langle s \rangle \times t \rangle, \langle s \times \langle t \rangle \rangle)$ and 4) $(\langle \langle s \rangle \times t \rangle, \langle \langle s \rangle \rangle \times t)$ and their symmetric analogs. The terms to which both components of the critical pairs rewrite are $\langle \{t\} \rangle$ for 1), $\langle t \times s \rangle$ for 2) and $\langle s \times t \rangle$ for 3) and 4). Thus, the rewrite system is Church-Rosser and therefore has unique normal forms.

The proof of the last statement is by induction on the structure of a given type. We limit ourselves only to types containing $\langle\cdot\rangle$. The base case is immediate. In the general case, consider three subcases: 1) $t = t_1 \times t_2$, 2) $t = \{t_1\}$, 3) $t = \langle t_1\rangle$. In subcase 1, $t' = t'_1 \times t'_2$, hence, if both t_1 and t_2 contain or-sets, $nf(t_1) = \langle t'_1\rangle, nf(t_2) = \langle t'_2\rangle$ and $t \twoheadrightarrow \langle t'_1\rangle \times \langle t'_2\rangle \twoheadrightarrow \langle t'_1 \times t'_2\rangle = \langle t'\rangle$ which is a normal form. Thus, $nf(t) = \langle t'\rangle$. The simple proofs of other cases are omitted. $\Box$

Having defined the rewrite rules for types, we must show how to apply these rules to objects. First, associate the following morphisms with the first three rules of the rewrite system:

$$or\_\rho_2 : t \times \langle s\rangle \longrightarrow \langle t \times s\rangle \qquad or\_\rho_1 : \langle t\rangle \times s \longrightarrow \langle t \times s\rangle \qquad or\_\mu : \langle\langle t\rangle\rangle \longrightarrow \langle t\rangle$$

One may be tempted to associate α with the rewrite rule $\{\langle t\rangle\} \longrightarrow \langle\{t\}\rangle$. However, the following subtlety prevents us from doing so. In the process of applying the functions corresponding to the rewrite rules, we may obtain an object of type $\{\langle t\rangle\}$ in which two or-sets, say $\langle a, b\rangle$, coincide. Using the set semantics forces us to keep only one copy. Hence, the set $\{a, b\}$ will not be included in the result (because it is not possible to choose a from one copy of $\langle a, b\rangle$ and b from the other, as we only have one copy), and consequently some of the objects can be lost from the conceptual level representation. This suggests keeping track of duplicates in order to obtain the correct result. In other words, we should use multisets rather than sets.

To formalize this, we introduce the new type $\{\!|t|\!\}$ of multisets of type t. Multiset types will only be used internally for the normalization process and should not be considered as a part of the language. With each type t we associate a type $t^{\mathbf{d}}$, which is obtained from t by replacing all set brackets $\{\}$ with multiset brackets $\{\!|\,|\!\}$ ($\mathbf{d}$ is for "duplicates"). Also, each object o of type t is turned into an object $o^{\mathbf{d}} : t^{\mathbf{d}}$ by making all sets into multisets with single multiplicities. Conversely, for every type t we define $t^{\mathbf{s}}$ by replacing all $\{\!|\,|\!\}$ with $\{\}$, and for every $o : t$, the object $o^{\mathbf{s}} : t^{\mathbf{s}}$ is defined by removing duplicates from all multisets and making them ordinary sets.

We need two operations on multisets that resemble two operations of *or-$\mathcal{NRA}$*. One is $\mathbf{d}\_map(f) : \{\!|s|\!\} \rightarrow \{\!|t|\!\}$, provided f is of type $s \rightarrow t$. It applies f to all elements of its input. Since no duplicates are removed, $\mathbf{d}\_map$ always preserves cardinality. The other operation is $\alpha_{\mathbf{d}} : \{\!|\langle t\rangle|\!\} \rightarrow \langle\{\!|t|\!\}\rangle$. It is defined exactly as α, except that its input may have duplicates, and duplicates are not removed from the result. For example, $\alpha_{\mathbf{d}}(\{\!|\langle 1, 2\rangle, \langle 1, 2\rangle|\!\}) = \langle\{\!|1, 1|\!\}, \{\!|1, 2|\!\}, \{\!|2, 2|\!\}\rangle$.

For types of the form $t^{\mathbf{d}}$ we define a rewrite system simliar to the one above, except that $\{\!|\langle t\rangle|\!\} \rightarrow \langle\{\!|t|\!\}\rangle$ is used in the place of $\{\langle t\rangle\} \rightarrow \langle\{t\}\rangle$. This rewrite system is also terminating and Church-Rosser. Moreover, for any ordinary type t, $nf(t)$ may be obtained as $t_0^{\mathbf{s}}$ where t_0 is the normal form of $t^{\mathbf{d}}$ in the new rewrite system. The functions we associate with the rules not involving bags are those shown above. We associate $\alpha_{\mathbf{d}}$ with $\{\!|\langle t\rangle|\!\} \rightarrow \langle\{\!|t|\!\}\rangle$.

Let t be a type and p a position in the derivation tree for t such that applying a rewrite rule with its associated function f to t at p yields type s. (We assume that t does not use $\{\cdot\}$.)

Our aim is to define a function $\mathsf{d\_app}(t, p, f) : t \to s$ showing the action of the rewrite rules on objects. Define it by induction on the structure of t:

- if p is the root of the derivation of t, then $\mathsf{d\_app}(t, p, f) = f$;
- if $t = t_1 \times t_2$ and p is in t_1 , then $\mathsf{d\_app}(t, p, f) = (\mathsf{d\_app}(t_1, p, f) \circ \pi_1, \pi_2)$;
- if $t = t_1 \times t_2$ and p is in t_2, then $\mathsf{d\_app}(t, p, f) = (\pi_1, \mathsf{d\_app}(t_2, p, f) \circ \pi_2)$;
- if $t = \{\!|t'|\!\}$ then $\mathsf{d\_app}(t, p, f) = \mathbf{d}\_map(\mathsf{app}(t', p, f))$;
- if $t = \langle t' \rangle$ then $\mathsf{app}(t, p, f) = or\_map(\mathsf{app}(t', p, f))$.

Given a type t and a rewriting strategy $r := t \xrightarrow{f_1} t_1 \xrightarrow{f_2} \ldots \xrightarrow{f_n} t_n$ such that the rewrite rule with associated function f_i is applied at a position p_i, we can extend the function $\mathsf{d\_app}$ to $\mathsf{d\_app}(t, r) : t \to t_n$ by $\mathsf{d\_app}(t, r) = \mathsf{d\_app}(t_{n-1}, p_n, f_n) \circ \ldots \circ \mathsf{d\_app}(t_1, p_2, f_2) \circ \mathsf{d\_app}(t, p_1, f_1)$.

Now assume that we are given an object x of type t. Suppose r is a rewriting that rewrites t to $nf(t)$. Associated with r, there is a rewriting $r^{\mathbf{d}}$ from $t^{\mathbf{d}}$ to $nf(t)^{\mathbf{d}}$. We define $\mathsf{app}(t, r) : t \to nf(t)$, the result of applying r, as follows:

$$\mathsf{app}(t, r)(x) \;=\; [\mathsf{d\_app}(t^{\mathbf{d}}, r^{\mathbf{d}})(x^{\mathbf{d}})]^{\mathbf{s}}$$

In other words, turn x into an object with multisets, apply $r^{\mathbf{d}}$ and then remove duplicates. The following theorem, which is a key result that allows us to introduce techniques for conceptual querying, says that the result of $\mathsf{app}(t, r)$ is independent of the rewriting r.

Theorem 4.2 (Coherence) *Given a type t, any two rewrite strategies $r_1, r_2 : t \twoheadrightarrow nf(t)$ yield the same result on objects. That is, for any object x of type t, $\mathsf{app}(t, r_1)(x) = \mathsf{app}(t, r_2)(x)$.*

Proof. Let us first explain the strategy for proving the theorem. It suffices to prove $\mathsf{d\_app}(t^{\mathbf{d}}, r_1^{\mathbf{d}})(x^{\mathbf{d}}) = \mathsf{d\_app}(t^{\mathbf{d}}, r_2^{\mathbf{d}})(x^{\mathbf{d}})$. We define an abstract rewrite system on objects of form $o^{\mathbf{d}}$ (that is, objects using multisets but not ordinary sets) by letting $x \to y$ iff y can be obtained from x by an application of one of the rewrite rules for types to x (by means of $\mathsf{d\_app}$). For instance, $(1, \langle\langle 1\rangle, \langle 2\rangle\rangle) \to (1, \langle 1, 2\rangle)$ by applying $\langle\langle t\rangle\rangle \to \langle t\rangle$ in the second position. If x is of type t and y is of type s, then $t \to s$ according to the rewrite system for types. Moreover, normal forms for our new rewrite system are precisely objects whose types are in normal form. Therefore, the rewrite system is terminating according to Proposition 4.1.

Now our goal is to prove that the new rewrite system is weakly Church-Rosser. That is, if x can be rewritten in one step to two objects, x_1 and x_2, then there exists an object x' such that both x_1 and x_2 can be rewritten to x' in zero or more steps. Then, by Newman's lemma [8], it will follow that the system is Church-Rosser and has unique normal forms, thus proving the theorem.

To prove weak Church-Rosserness, assume that $x \to x_1$ by means of rule r_1 in position p_1 in t and $x \to x_2$ by means of rule r_2 in position p_2 in t. We denote the functions that correspond to applying r_1 and r_2 by f_1 and f_2 respectively. Notice that if positions p_1 and p_2 are in two different subtrees determined by a pair formation, then the existence of x' is immediate. Hence, we can assume that one position, say p_1, is closer to the root than p_2 because $\{\!|\cdot|\!\}$ and $\langle\cdot\rangle$ are unary type constructors. The rest of the proof is a straightforward case analysis. We present two cases for illustration.

The rule applied in p_1 is $s \times \langle t\rangle \to \langle s \times t\rangle$, and p_2 occurs inside the tree for s. The object therefore is a pair (x_1, x_2) and the function applied is or_ρ_2. Assume that $\mathsf{d\_app}(s, p_2, r_2)(x_1) = x_1'$. Then we obtain

$$\begin{tikzcd}
 & \langle (x_1, x_2^i) \mid x_2^i \in x_2\rangle \arrow[dr, "or_map(f_2 \circ \pi_1, \pi_2)"] & \\
(x_1, x_2) \arrow[ur, "or_\rho_2"] \arrow[dr, "(f_2 \circ \pi_1, \pi_2)"'] & & \langle (x_1', x_2^i) \mid x_2^i \in x_2\rangle \\
 & (x_1', x_2) \arrow[ur, "or_\rho_2"'] &
\end{tikzcd}$$

The rule applied in p_1 is $\{\!|\langle t\rangle|\!\} \to \langle\{\!|t|\!\}\rangle$, and p_2 is inside t. The object therefore is a bag of or-sets $\mathcal{X} = \{\!|X_1, \ldots, X_p|\!\}$ where $X_i = \langle x_1^i, \ldots, x_{n_i}^i\rangle$ and the function applied is $\alpha_{\mathbf{d}}$. Assume that applying f_2 to every x_j^i yields y_j^i. The result of applying f_2 (by means of $\mathsf{d\_app}$) to $\mathcal{X}$ is $\mathcal{Y} = \{\!|\langle y_j^i \mid j = 1, \ldots, n_i\rangle \mid i = 1, \ldots, p|\!\}$. Consider the following diagram, in which we do not yet say what the target is. (We only note that applying $or\_map(\mathbf{d}\_map(f_2)) \circ \alpha_{\mathbf{d}}$ and $\alpha_{\mathbf{d}} \circ \mathbf{d}\_map(or\_map(f_2))$ to $\mathcal{X}$ yield objects of the same type.)

$$
\begin{tikzcd}
 & \langle \{\!|x^i_{h(i)} \mid i = 1, \ldots, p|\!\} \mid h \in \mathcal{F}_{\mathcal{X}} \rangle \arrow[dr, "\mathit{or\_map}(\mathbf{d}\_\mathit{map}(f_2))"] & \\
\mathcal{X} \arrow[ur, "\alpha_{\mathbf{d}}"] \arrow[dr, "\mathbf{d}\_\mathit{map}(\mathit{or\_map}(f_2))"'] & & \bullet \\
 & \{\!|\langle y^i_j \mid j = 1, \ldots, n_i \rangle \mid i = 1, \ldots, p|\!\} \arrow[ur, "\alpha_{\mathbf{d}}"'] &
\end{tikzcd}
$$

To prove the case, we must show that this diagram commutes. First notice that because of the semantics of $\mathbf{d}\_\mathit{map}$, $\mathcal{Y}$ has p elements. Denote $\langle y^i_j \mid j = 1, \ldots, n_i \rangle$ by Y_i. First assume that $B \in \mathit{or\_map}(\mathbf{d}\_\mathit{map}(f_2)) \circ \alpha_{\mathbf{d}}(\mathcal{X})$. Then for some $h \in \mathcal{F}_{\mathcal{X}}$ we have $B = \{\!|f_2(x^i_{h(i)}) \mid i = 1, \ldots, p|\!\}$. Assume that y^i_j is such that $y^i_j = f_2(x^i_{h(i)})$, and define $g \in \mathcal{F}_{\mathcal{Y}}$ by letting $g(i)$ be j. Then $B = \{\!|y^i_{g(i)} \mid i = 1, \ldots, p|\!\}$ and hence $B \in \alpha_{\mathbf{d}}(\mathcal{Y}) = \alpha_{\mathbf{d}} \circ \mathbf{d}\_\mathit{map}(\mathit{or\_map}(f_2))(\mathcal{X})$.

Conversely, let $B \in \alpha_{\mathbf{d}}(\mathcal{Y})$. Then for some $g \in \mathcal{F}_{\mathcal{Y}}$ we have $B = \{\!|y^i_{g(i)} \mid i = 1, \ldots, p|\!\}$. Since $y^i_{g(i)} \in Y_i = \mathit{or\_map}(f_2)(X_i)$, there exists $x^i_j \in X_i$ such that $f_2(x^i_j) = y^i_{g(i)}$. Define $h \in \mathcal{F}_{\mathcal{X}}$ by letting $h(i) = j$ for all i. Then $B = \{\!|f_2(x^i_{h(i)}) \mid i = 1, \ldots, p|\!\}$ and therefore $B \in \mathit{or\_map}(\mathbf{d}\_\mathit{map}(f_2)) \circ \alpha_{\mathbf{d}}(\mathcal{X})$. Hence, $\mathit{or\_map}(\mathbf{d}\_\mathit{map}(f_2)) \circ \alpha_{\mathbf{d}}(\mathcal{X}) = \alpha_{\mathbf{d}} \circ \mathbf{d}\_\mathit{map}(\mathit{or\_map}(f_2))(\mathcal{X})$, which shows that the diagram commutes and this proves the case. □

Therefore, all objects with the same meaning at the conceptual level rewrite to the same normal form. The intuitive notion of the conceptual meaning can now be rigorously defined as the normal form. We define the *conceptual query language* or-$\mathcal{NRA}^+$ by adding the new construct

$$\frac{}{\mathit{normalize}^t : t \to \mathit{nf}(t)}$$

to or-$\mathcal{NRA}$. By the coherence theorem, $\mathit{normalize}^t$ can be implemented as $\mathsf{app}(t, r)$ where r is any rewriting $t \longrightarrow\!\!\!\!\!\rightarrow \mathit{nf}(t)$. Conceptual *queries* are now defined as queries on normal forms. That is, queries of form $q \circ \mathit{normalize}$, where q is a structural query.

In the remainder of this section, we show that for each type t, it is possible to express $\mathit{normalize}^t$ in or-$\mathcal{NRA}$. That is, the introduction of $\mathit{normalize}^t$ to build the conceptual language is a matter of convenience. Nonetheless, it is important to include $\mathit{normalize}^t$ in the conceptual language because it cannot be defined in a polymorphic way.

Corollary 4.3 *For each type t, $\mathit{normalize}^t$ is expressible in or-$\mathcal{NRA}$.*

Proof sketch. As it follows from the proof of the coherence theorem, to express $normalize^t$ we have to simulate the operations $\alpha_{\mathbf{d}}$ and $\mathbf{d}\_map$ in a way that retains duplicates. We do it as follows. First, define a family of translations of types $(\cdot)'$: $b' = b$, $(s \times t)' = s' \times t'$, $\langle s \rangle' = \langle s' \rangle$, $\{s\}' = \{s' \times u_s\}$. Now each object $o : t$ is translated into an object $o' : t'$. The only nonobvious case in the translation is the set case: $\{x_1, \ldots, x_n\}' = \{(x'_1, y_1), \ldots, (x'_n, y_n)\}$, where all y_is are distinct. One such translation is definable in *or-*$\mathcal{NRA}$: u_s in the set case is taken to be s, and $\{x_1, \ldots, x_n\}' = \{(x'_1, x_1), \ldots, (x'_n, x_n)\}$. That is, each element of a set gets a unique tag.

Now, for each function $f : s_1 \to s_2$ used in the process of normalization, we define a new function $f' : s'_1 \to s'_2$ that simulates the action of f using tags in sets to retain duplicates. We only need to consider the case of α and map. For $\alpha : \{\langle s \rangle\} \to \langle \{s\} \rangle$ and $\{\langle s \rangle\}' = \{\langle s' \rangle \times u\}$, define $\alpha' = \alpha \circ map(or\_\rho_1) : \{\langle s' \rangle \times u\} \to \langle \{s' \times u\} \rangle$. For $g : s' \to t'$, define $map(g)'$ as $map((g' \circ \pi_1), \pi_2)$. Finally, let o be an object of type t. Translate it to $o' : t'$ as shown above, and simulate the normalization process using α' in the place of $\alpha_{\mathbf{d}}$ and map' in the place of $\mathbf{d}\_map$. At the end, project out all tags. Now it is an easy application of the coherence theorem to show that the object thus obtained is $normalize^t(o)$. $\Box$

Two questions can be asked of this new query language. First, how much information is lost by normalization? There are different objects that normalize to the same one and information from the structural level could be lost. Second, how costly is normalization? We address these problems in subsequent sections. In the next section it is shown that normalization is often *lossless*, i.e. many queries are unaffected by the loss of structural information. In Section 6, upper bounds for the size of normalized objects are found.

5 Losslessness of Normalization

This section investigates whether the process of normalization loses anything "that can be regarded as critical." If loss of information is inevitable in the general case, then one would like to obtain a set of general sufficient (and if possible, necessary) conditions that guarantee losslessness of normalization. In order to proceed, a criterion on what normalization can be regarded as "losing nothing essential" has to be formulated.

Definition. *Given a definable morphism $f : s \to t$. Suppose there is a morphism $preserve(f) : nf(\langle s \rangle) \to nf(\langle t \rangle)$ such that $preserve(f) \circ normalize^{\langle s \rangle} \circ or\_\eta^s = normalize^{\langle t \rangle} \circ or\_\eta^t \circ f$, provided the input is restricted to objects not containing any empty or-set. Then normalization is* lossless *with respect to f.*

Let us first justify the definition given above. The proviso on the input is necessary because all objects containing empty or-set have the same normal form, namely $\langle \rangle$. Recalling that $\langle \rangle$ stands for inconsistency, such objects are conceptually inconsistent and should be omitted. The use of or_η^s and or_η^t is a technical device to ensure that the normal forms produced always look like $\langle d_1, ..., d_n\rangle$ where $d_1, .., d_n$ have no or-sets. This is justified since $or\_\eta\ d$ is conceptually d for any d.

The equation itself is justified because $preserve(f)$ takes the meaning of the input to f and returns the meaning of the output of f.

It turns out that it is not easy to achieve losslessness of normalization with respect to an arbitrarily given morphism f. There is no simple method to discover the required $preserve(f)$. However, we have been able to isolate the morphisms that can give rise to possible difficulty.

Theorem 5.1 (Losslessness) *Let $f : s \to t$ be a morphism of or-$\mathcal{NRA}$ not containing any $K\langle\rangle$; p where some or-set appears in $Type(p)$; $\rho_2^{u,v}$, μ^u, and $\cup^u$ where u has some or-sets; $map(g) : \{u\} \to \{v\}$ where u or v have some orsets; and $(g, h) : r \to u \times v$ where r, u, or v have some or-sets. Then normalization is lossless with respect to f. Moreover, the $preserve(f)$ that makes normalization lossless has a map-like property and preserves consistency. That is, $preserve(f) = or\_\mu \circ or\_map(preserve(f) \circ or\_\eta)$ and $preserve(f)(x) \neq \langle\rangle$ whenever x contains no $\langle\rangle$.*

Proof. Let $preserve\ t$ be the type obtained by converting every base type b in t to $\langle b\rangle$. Let $preserve_t : t \to preserve\ t$ be the morphism such that $preserve_t(x)$ is the object obtained by mapping every base type object $o : b$ in x to the singleton or-set $\langle o\rangle$. Using the fact that normalization is coherent, it is easy to show by induction on t that $normalize \circ or\_\eta^t = normalize \circ preserve_t$. Consequently, to prove the theorem, we can instead prove the commutativity of

$$\begin{tikzcd}
x : s \arrow[r, "preserve_s"] \arrow[d, "f"'] & \bullet : s' \arrow[r, "normalize"] & \bullet : \langle s''\rangle \arrow[d, "preserve(f)"] \\
\bullet : t \arrow[r, "preserve_t"'] & \bullet : t' \arrow[r, "normalize"'] & \bullet : \langle t''\rangle
\end{tikzcd}$$

for any complex object $x : s$ having no empty or-set and any morphism $f : s \to t$ satisfying the preconditions of the theorem, where $preserve(f)$ is defined by structural induction on f below.

Case f is id. Then $preserve(f) = id$.

Case f is η, π_1, π_2, μ, $K\{\}$, Kc, $!$, $\cup$, ρ_2, or p. Then $preserve(f) = or\_map(f)$.

Case f is (g, h). Then $preserve(g, h) = or\_\mu \circ or\_map(or\_\rho_1) \circ or\_\rho_2 \circ (preserve\ g, preserve\ h)$.

Case f is $g \circ h$. Then $preserve(g \circ h) = preserve(g) \circ preserve(h)$.

Case f is $map(g)$. Then $preserve(map\ g) = or\_\mu \circ or\_map(\alpha) \circ or\_map(map(preserve(g) \circ or\_\eta))$.

Case f is α, or_η, or_ρ_2, or or_μ. Then $preserve(f) = id$.

Case f is $or\_\cup$. Then $preserve(f) = or\_\mu \circ or\_map(or\_\cup \circ (or\_\eta \times or\_\eta))$.

Case f is $or\_map(g)$. Then $preserve(or\_map(g)) = preserve(g)$.

It is readily verified that $preserve(f)$ is map-like and preserves consistency. The proof that the diagram commutes is by induction on f and uses the coherence theorem in several places. We present two illustrative cases.

Suppose f is $or\_map(g)$, where $g : u \to v$. Then $s = \langle u \rangle$ and $t = \langle v \rangle$. By hypothesis, $preserve(g)$ exists and is map-like. Now consider the diagram below.

$$\begin{tikzcd}
x : \langle u \rangle \arrow[r, "preserve"] \arrow[d, "or_map(g)"] & \bullet : \langle u' \rangle \arrow[r, "or_map(normalize)"] & \bullet : \langle\langle u'' \rangle\rangle \arrow[r, "or_\mu"] \arrow[d, "or_map(preserve\ g)"'] & \bullet : \langle u'' \rangle \arrow[d, "preserve\ g"'] \\
\bullet : \langle v \rangle \arrow[r, "preserve"'] & \bullet : \langle v' \rangle \arrow[r, "or_map(normalize)"'] & \bullet : \langle\langle v'' \rangle\rangle \arrow[r, "or_\mu"'] & \bullet : \langle v'' \rangle
\end{tikzcd}$$

The left rectangle commutes by hypothesis. The right rectangle commutes because $preserve(g)$ is map-like. Hence the entire diagram commutes. By the coherence theorem, $normalize^{\langle u' \rangle} = or\_\mu^{u''} \circ or\_map(normalize^{u'})$ and $normalize^{\langle v' \rangle} = or\_\mu^{v''} \circ or\_map(normalize^{v'})$. So the original diagram commutes and the case follows.

Suppose f is $\pi_1^{u,v}$. Then $s = u \times v$ and $t = u$. Let $or\_cp = or\_\mu \circ or\_map(or\_\rho_1) \circ or\_\rho_2$. Consider the diagram below.

$$\begin{tikzcd}
x : u \times v \arrow[r, "preserve"] \arrow[d, "\pi_1"'] & \bullet : u' \times v' \arrow[r, "(normalize \circ \pi_1, normalize \circ \pi_2)"] \arrow[d, "\pi_1"'] & y : \langle u'' \rangle \times \langle v'' \rangle \arrow[r, "or_cp"] \arrow[d, "\pi_1"'] & \bullet : \langle u'' \times v'' \rangle \arrow[d, "or_map(\pi_1)"'] \\
\bullet : u \arrow[r, "preserve"'] & \bullet : u' \arrow[r, "normalize"'] & \bullet : \langle u'' \rangle \arrow[r, "id"'] & \bullet : \langle u'' \rangle
\end{tikzcd}$$

The two left rectangles obviously commute. By assumption, x has no empty or-set. Thus y has no empty or-set. Therefore, the right rectangle commutes. Hence the whole diagram commutes. Finally, the coherence theorem is applied to conclude the case. $\square$

Since p is generally an uninterpreted primitive, the qualification that $Type(p)$ has no or-set is necessary. This means that equality tests $=^t$, where t has or-sets, have been excluded. $=^t$ is an equality test that is structural. Normalization is a process that removes structural differences from objects that are conceptually identical. Hence one cannot expect normalization to be lossless with respect to $=^t$.

On the other hand, the restrictions placed on μ, $\cup$, and $map(g)$ can be lifted under certain circumstances. Recall from Corollary 4.3 that in order to express *normalize* in *or-$\mathcal{NRA}$*, elements of sets are tagged with unique identifiers. This tagging is to prevent or-sets from being collapsed prematurely. The problem with μ, $\cup$, and $map(g)$ is that they can collapse two or-sets into one. There are two solutions to this problem. The first is to make sure that these operations are not applied to objects involving or-sets, as required by the preconditions of the losslessness theorem. The second is to make sure that these operations are not applied to objects in which or-sets can be collapsed. For example, if it is known that $\cup$ is only applied to a pair of sets of or-sets having empty intersection, then we can still achieve losslessness for $\cup$ using $preserve(\cup)$ as given above.

Given an *or-$\mathcal{NRA}$* morphism $f : s \to t$ and an object $x : s$ containing some or-sets. Then x conceptually represents several values $x_1, ..., x_n$. Suppose $f\ x$ is an object containing or-sets; then it conceptually represents several values $y_1, ..., y_m$. It is desirable to discover which one of $x_1, ..., x_n$ leads to which one of $y_1, ..., y_m$. This is a question of searching for a *conceptual analog* of f that associates each x_i in *normalize* x to a subset of *normalize*$(f\ x)$.

The idea of a conceptual analog of a morphism is illustrated in Figure 2. One would like to know which combination of the conceptual values of the input gives rise to which subset of the conceptual values of the output. However, the ideal situation can only be approximated. As a first attempt, for each possible conceptual value x_i of the input x, we aim only to account for some of the conceptual values in the output that are due to it. Some conceptual values y_j in the output may be left unaccounted for. For example, the last element of *normalize* y in the figure. Similarly, the picture given for each input x_i is only partial. For example, the second element of *normalize* x in the figure might in reality contribute to 3 values in the output but the conceptual analog discovers only 2. This approximation to conceptual analog is formalized below.

Definition. *Let $f : s \to t$ be a definable morphism of or-$\mathcal{NRA}$. Then its* conceptual analog *is a morphism $preserve(f)$ such that for all $x : s$, $(preserve(f) \circ normalize^{\langle s \rangle} \circ or\_\eta^s)(x)$ is included in $(normalize^{\langle t \rangle} \circ or\_\eta^t \circ f)(x)$.*

There is some relationship between losslessness and conceptual analog. A conceptual analog of f that accounts for every element in the output is a morphism that makes normalization lossless with respect to f. Hence the search for a lossless $preserve(f)$ can be generalized as a search for a conceptual analog of f that accounts for each possible conceptual value of the output.

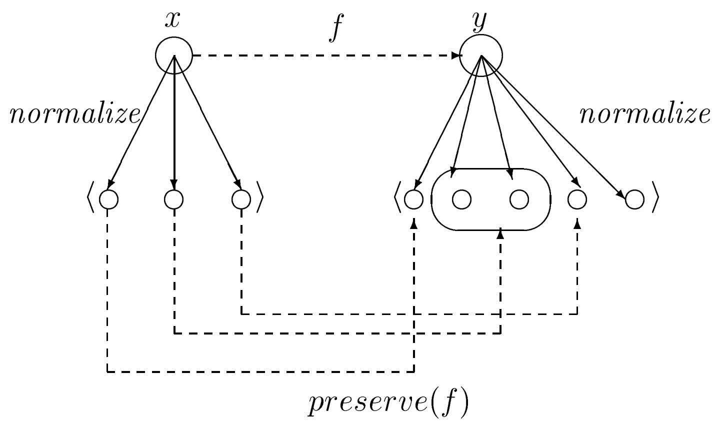

Figure 2: Conceptual analog of morphism f

The losslessness theorem and conceptual analog can be given a somewhat simpler description if types are restricted only to those containing or-sets. The morphism $preserve_t : t \to preserve\, t$ defined in the proof of Theorem 5.1 forces an object of type t into an object of type $preserve\, t$ by inserting singleton or-sets wherever possible. Types of the form $preserve\, t$ can be described by the following grammar: $t ::= \langle b \rangle \mid t \times t \mid \{t\} \mid \langle t \rangle$. We call such a type a *pure or-type.* It can be easily seen that any object x is conceptually equivalent to $preserve(x)$, provided x has or-sets.

Given an or-$\mathcal{NRA}$-morphism $f : s \to t$ and two objects $x : s$ and $y = f(x) : t$, let $normalize(x) = \langle x_1, \ldots, x_n \rangle$ and $normalize(y) = \langle y_1, \ldots, y_m \rangle$, $nf(s) = \langle s' \rangle$ and $nf(t) = \langle t' \rangle$. Our motivation to study losslessness was to find a conceptual analog of f. What can such an analog be? As the first approximation, it is given by a function $f' : s' \to \langle t' \rangle$ which associates with each element x_i in $normalize(x)$ a subset of $normalize(y)$, thus defining the action of f on elements its input could possibly stand for. The morphism $preserve(f) : nf(s) \to nf(t)$ can now be defined as $or\_\mu \circ or\_map(f')$.

How could one refine the action of f on elements of normalized object? There are two ways to do so. First, one can require that this action be defined unambiguously. That is, f' maps every element from $normalize(x)$ into a unique element of $normalize(y)$, thus having type $s' \to t'$. $preserve(f)$ can then be reconstructed as $or\_map(f')$. Second, one can require that all the elements of $normalize(y)$ be accounted for. That is, $preserve(f) \circ normalize(x) = normalize(y)$. In other words, $preserve(f)$ is onto. It is not hard to see that this is precisely the definition of losslessness in the case of pure or-types.

Proposition 5.2 *Let s and t be pure or-types. Let $f : s \to t$ be a morphism in* or-$\mathcal{NRA}$ *that does not use any primitive p where $Type(p)$ has or-sets and any μ^u, $\cup^u$, or $map(g) : \{u\} \to \{v\}$ where u or v involve or-sets. Then there exists a conceptual analog $preserve(f)$ for f. This* *conceptual analog is map-like. However, if f does not use $K\langle\rangle$, or_$\cup$ and $(\cdot,\cdot)$, then it can be given the form or_map$(\cdot)$. Moreover, if f does not use $K\langle\rangle$, $(\cdot,\cdot)$, and ρ_2, then it is also onto with respect to input having no $\langle\rangle$.*

Proof sketch. The precondition is weaker than that of Theorem 5.1 because we merely required that $(\mathit{preserve}(f) \circ \mathit{normalize} \circ \mathit{or\_\eta})(x)$ be included in $\mathit{normalize} \circ \mathit{or\_\eta} \circ f)(x)$, as opposed to being equal. The proof is a simple adaptation of the proof of Theorem 5.1. After defining $\mathit{preserve}(K\langle\rangle)$ as $\mathit{or\_\mu} \circ \mathit{or\_map}(K\langle\rangle \circ !)$, we can define the remaining $\mathit{preserve}(f)$ the same way as in Theorem 5.1. This makes $\mathit{preserve}(f)$ map-like.

There are only three cases in which $\mathit{preserve}(f)$ as defined above cannot be made into the form $\mathit{or\_map}(\cdot)$: $K\langle\rangle$, $(\cdot,\cdot)$, and $\mathit{or\_}\cup$. We illustrate the case $f = \mathit{or\_}\cup : \langle t\rangle \times \langle t\rangle \to \langle t\rangle$. To see why the translation cannot be of the form $\mathit{or\_map}(\cdot)$, let t be a base type, say *int*, and consider an object $x = (\langle 1,2\rangle, \langle 3\rangle)$. Applying $\mathit{normalize} \circ \mathit{or\_}\cup$ gives $\langle 1,2,3\rangle$ while applying *normalize* yields $\langle (1,3),(2,3)\rangle$. Clearly no mapping over the latter object can produce the former.

The preconditions of this proposition is weaker than that of Theorem 5.1 in three places: f can use arbitrary $K\langle\rangle$, $(\cdot,\cdot)$, and ρ_2. It is precisely these three operations that may destroy the surjectivity of $\mathit{preserve}(f)$. We illustrate the case $f = \rho_2 : s \times \{t\} \to \{s \times t\}$. Since s and t are pure or-types, $\mathit{nf}(s) = \langle s'\rangle$ and $\mathit{nf}(t) = \langle t'\rangle$. Then $\mathit{preserve}(\rho_2)$ must have type $\langle s' \times \{t'\}\rangle \to \langle \{s' \times t'\}\rangle$. We take $\mathit{preserve}(\rho_2)$ to be $\mathit{or\_map}(\rho_2^{s',t'})$. An easy application of the coherence theorem shows that for any object x of type $s \times \{t\}$, $\mathit{or\_map}(\rho_2^{s',t'}) \circ \mathit{normalize}(x) \subseteq \mathit{normalize} \circ \rho_2^{s,t}(x)$. So $\mathit{preserve}(\rho_2)$ is a conceptual analog of ρ_2. To see that it is not onto, let x be $(\langle 1,2\rangle, \{3,4\})$. Then $\mathit{preserve}(\rho_2)(\mathit{normalize}(x)) = \langle \{(1,3),(1,4)\}, \{(2,3),(2,4)\}\rangle$. On the other hand, $\mathit{normalize}(\rho_2(x)) = \langle \{(1,3),(1,4)\}, \{(1,3),(2,4)\}, \{(2,3),(1,4)\}, \{(2,3),(2,4)\}\rangle$. $\square$

6 Costs of Normalization

We have seen before that the complexity of or-$\mathcal{NRA}^+$ queries can be exponential. In particular, the cardinality of $\mathit{normalize}(x)$ can be exponential in the size of x provided that α was used in the course of normalization. In fact, the example given in Section 2 shows that even one application of α may result in an or-set of exponential cardinality. If one tries to estimate the cost of normalization by "brute force," a hyperexponential upper bound can be immediately obtained: indeed, if n is the size of x, applying the costly α $O(n)$ times seems to yield a hyperexponential bound.

In this section we show that the fear of hyperexponentiality is not justified. In fact, both the cardinality of $\mathit{normalize}(x)$ and its size are in the worst case exponential in the size of x. The first result in this section explains why consecutive applications of α still yield objects of exponential size. Then we proceed to find upper bounds on the cardinality and the size of normalized objects. The last result in this section shows that there exist existential queries involving normalization which cannot be evaluated in polynomial time.

Let x be an object and $y = \mathit{normalize}(x)$. Define $m(y)$ as the number of elements in y if it is an or-set and 1 otherwise. Uniformly, $m(x) = |\mathit{normalize}(\mathit{or\_}\eta(x))|$. The *size* of an object is defined inductively: the size of an atomic object is 1, $\mathsf{size}\,(x, y) = \mathsf{size}\,x + \mathsf{size}\,y$, $\mathsf{size}\,\{x_1, \ldots, x_n\}$ $= \mathsf{size}\,\langle x_1, \ldots, x_n\rangle = \mathsf{size}\,x_1 + \ldots + \mathsf{size}\,x_n$.

To work with objects, it is convenient to associate rooted labeled trees with them. A tree $\mathcal{T}x$ associated with an atomic object x is defined as a one-node tree labeled by x. $\mathcal{T}(x, y)$ is a tree with the root labeled by $\times$, and two subtrees rooted at its children are $\mathcal{T}x$ and $\mathcal{T}y$. $\mathcal{T}\{x_1, \ldots, x_n\}$ (or $\mathcal{T}\langle x_1, \ldots, x_n\rangle$) is a tree whose root is labeled by $\{\}$ (or $\langle\rangle$) and n subtrees rooted at its children are $\mathcal{T}x_1, \ldots, \mathcal{T}x_n$. In view of this definition, $m(x)$ can be redefined as the number of children of the root of $\mathcal{T}\mathit{normalize}(x)$ if the root is labeled by $\langle\rangle$ and 1 otherwise. $\mathsf{size}\,x$ is the number of leaves in $\mathcal{T}x$.

Intuitively, the following proposition says that the "internal" structure of $\mathcal{T}x$ does not contribute to the creation of new possibilities in $\mathit{normalize}(x)$, and the number of such possibilities $m(x)$ is determined by the or-sets which are closest to the leaves.

Proposition 6.1 *Let x be an object, and $v_1, \ldots, v_k$ be the nodes in $\mathcal{T}x$ labeled by $\langle\rangle$, such that the subtrees rooted at v_i's do not have other nodes labeled by $\langle\rangle$ (i.e. they are or-sets closest to the leaves). Let m_i be the number of children of v_i, $i = 1, ..., k$. Then, if $k \neq 0$,*

$$m(x) \leq \prod_{i=1}^{k}(m_i + 1)$$

Proof. We proceed by induction on the structure of the object x. We consider only objects containing or-sets. The base case (i.e. or-sets of objects of base types) is obvious. Let $x = (x_1, x_2)$. Assume that both x_1 and x_2 contain or-sets and $v_1, \ldots, v_p$ are nodes of $\mathcal{T}x_1$ and $v_{p+1}, \ldots, v_k$ are nodes of $\mathcal{T}x_2$. Then, by the induction hypothesis, $m(x_1) \leq \prod_{i=1}^{p}(m_i + 1)$ and $m(x_2) \leq \prod_{i=p+1}^{k}(m_i + 1)$. By coherence, $\mathit{normalize}(x) = \mathit{or\_}\rho(\mathit{normalize}(x_1), \mathit{normalize}(x_2))$ where $\mathit{or\_}\rho$ pairs each item in its first argument with each item in its second argument (it can be easily expressed in *or-$\mathcal{NRA}$*). Therefore, $m(x) \leq m(x_1)m(x_2) \leq \prod_{i=1}^{k}(m_i + 1)$. Two other cases when either x_1 or x_2 contains or-sets are similar.

Let $x = \{x_1, \ldots, x_n\}$. Then all x_i's contain or-sets. Again, by coherence,

$$\mathit{normalize}(x) = [\alpha_{\mathbf{d}}(\{\!|\mathit{normalize}(x_1), \ldots, \mathit{normalize}(x_n)|\!\})]^{\mathbf{s}}$$

Therefore, $m(x) \leq \prod_{i=1}^{n} m(x_i)$ and the result follows from the induction hypothesis.

Finally, if $x = \langle x_1, \ldots, x_n \rangle$, there are two cases. If x_i's do not contain or-sets, then $m(x) = n \leq n + 1$. If they do contain or-sets, then by coherence

$$normalize(x) = or\_\mu(\langle normalize(x_1), \ldots, normalize(x_n) \rangle)$$

i.e. $m(x) \leq \Sigma_{i=1}^{n} m(x_i) \leq \prod_{i=1}^{n} m(x_i)$ because $m(\cdot) \geq 2$. The case now follows from the hypothesis. $\square$

This proposition explains why there is an exponential upper bound for $m(x)$ despite the fact that α can be applied many times. The following result finds a tight upper bound in terms of the size rather than the tree structure.

Theorem 6.2 *Let x be an object with $\mathsf{size}\, x = n$. Then*

$$m(x) \leq \sqrt[3]{3}^{\,n}$$

Moreover, for any n divisible by 3 there exists an object x such that $\mathsf{size}\, x = n$ and $m(x) = \sqrt[3]{3}^{\,n}$.

Proof. As in the proof of Proposition 6.1, consider only objects containing or-sets. Proceed by induction on the number of steps of normalization. If the object is already normalized, we are done. Assume $normalize(x)$ is obtained by one step of normalization. Then this step is one of the maps associated with the rewrite rules, so we have four cases. Notice that in the base cases we may assume w.l.o.g that any element of a set or an or-set is of base type since this will give us the maximal possible $m(x)$ for a given $\mathsf{size}\, x$.

Case 1. $x = (x_1, x_2)$ where $x_1 = \langle x_1^1, \ldots, x_{n-1}^1 \rangle$. Then $normalize(x) = or\_\rho_1(x)$ and it is an easy arithmetic exercise to show that $m(x) = n - 1 \leq \sqrt[3]{3}^{\,n}$.

Case 2 when or_ρ_2 is applied to obtain the normal form is similar.

Case 3. Let $x = \{X_1, \ldots, X_k\}$ where each X_i is an or-set $\langle x_1^i, \ldots, x_{k_i}^i \rangle$ where all x_j^i are elements of a base type. Since we are interested in an upper bound, assume w.l.o.g. that all x_j^i's are distinct (if they are not, some of sets in $normalize(x)$ could collapse). Let $X = \bigcup_{i,j} x_j^i$. Define a graph $G = (X, E)$ where $(x_{j_1}^{i_1}, x_{j_2}^{i_2})$ is in E iff $i_1 \neq i_2$. Let $normalize(x) = \alpha(x) = \langle Y_1, \ldots, Y_p \rangle$ (Y_k's are sets). Then it follows from the definition of α that $Y_1, \ldots, Y_p$ are precisely the cliques of G. Since $n = \mathsf{size}\, x = |X|$, applying the upper bound on the number of cliques for a graph with n vertices [28], we obtain $p = m(x) \leq \sqrt[3]{3}^{\,n}$.

Case 4. $x = \langle X_1, \ldots, X_k \rangle$ where X_i's are or-sets of a base type. Then $normalize(x) = or\_\mu(x)$ and $m(x) \leq n$. Again, simple arithmetic shows that $n \leq \sqrt[3]{3}^{\,n}$. Hence, $m(x) \leq \sqrt[3]{3}^{\,n}$.

The proof of the general case is very similar to the proof of Proposition 6.1 and we will show only one case. Let $x = \{x_1, \ldots, x_k\}$ where x_i's are not normalized. Then $normalize(x)$ is obtained by applying $\alpha_{\mathbf{d}}$ to $\{\!|normalize(x_1), \ldots, normalize(x_n)|\!\}$ and removing duplicates from the result. Let $\mathsf{size}\, x_i = n_i$. By the induction hypothesis, $m(x_i) \leq \sqrt[3]{3}^{n_i}$. We now have

$$m(x) \leq \prod_{i=1}^{k} m(x_i) \leq \prod_{i=1}^{k} \sqrt[3]{3}^{n_i} \leq \sqrt[3]{3}^{n}$$

The other cases are similar. To show the tightness of the upper bound, let $n = 3k, k > 0$. Assume that we have a base type whose domain is infinite (a typical example is *int*). Let $b_1, \ldots, b_n$ be n distinct elements of such a type. Let

$$x = \{\langle b_1, b_2, b_3\rangle, \langle b_4, b_5, b_6\rangle, \ldots, \langle b_{n-2}, b_{n-1}, b_n\rangle\}$$

Then $\mathsf{size}\, x = n$ and $normalize(x) = \alpha(x)$ which contains $3^k = \sqrt[3]{3}^{n}$ elements. The theorem is completely proved. $\Box$

Using Theorem 6.2, one can prove the following upper bound on the size of normal forms by induction on the steps of the normalization process.

Theorem 6.3 *Let x be an object with* $\mathsf{size}(x) = n$ *where $n > 1$. Then*

$$\mathsf{size}\, normalize(x) \leq \frac{n}{2} \sqrt[3]{3}^{n}$$

Proof. Similarly to the proof of Theorem 6.2, proceed by induction on the steps of normalization. We start with base cases, i.e. consider an application of or_ρ_2 or or_ρ_1 or α or or_μ.

Case 1. $x = (x_1, x_2)$ where $x_1 = \langle x_1^1, \ldots, x_k^1\rangle$. Let $\mathsf{size}\, x_1 = s_1$, $\mathsf{size}\, x_i^1 = \sigma_i$. Then $s_1 + \sigma_1 + \ldots + \sigma_k = n$. Since $normalize(x) = or\_\rho_1(x)$, $\mathsf{size}\, normalize(x) = ks_1 + \sigma_1 + \ldots + \sigma_k = ks_1 + (n - s_1) \leq (n - s_1)s_1 + n - s_1 \leq 2n - 2$. Since empty sets and or-sets are excluded, $n \geq 2$ in this case and therefore $2n - 2 \leq \frac{n}{2}\sqrt[3]{3}^{n}$.

Case 2 when or_ρ_2 is applied is similar.

Case 3. Let $x = \{X_1, \ldots, X_l\}$ where each X_i is an or-set $\langle x_1^i, \ldots, x_{k_i}^i\rangle$ where all x_j^i have types containing no or-set. Let $\mathsf{size}\, x_j^i = s_j^i$ and

$$\sum_{j=1}^{k_i} s_j^i = \sigma_i \qquad\qquad \sum_{i=1}^{l} \sigma_i = n$$

Then an easy calculation shows that $\mathsf{size}\, normalize(x) = \mathsf{size}\, \alpha(x)$ is bounded above by

$$\sigma_1 \cdot k_2 \cdot \ldots \cdot k_l + \sigma_2 \cdot k_1 \cdot k_3 \cdot \ldots \cdot k_l + \ldots + \sigma_l \cdot k_1 \cdot \ldots \cdot k_{l-1} \leq l \cdot \sigma_1 \cdot \ldots \cdot \sigma_l$$

Therefore, we need to maximize $l \cdot \sigma_1 \cdot \ldots \cdot \sigma_l$ under the constraint $\sigma_1 + \ldots + \sigma_l = n$. A standard argument shows that such a maximum is bounded above by

$$\begin{cases} 1 & \text{if } n = 1 \\ \frac{n}{2}\sqrt{2}^n & \text{if } 1 < n < 21 \\ \frac{n}{3}\sqrt[3]{3}^n & \text{if } n \geq 21 \end{cases}$$

If it easy to see that for $n > 1$, the upper bounds given above are less than $\frac{n}{2}\sqrt[3]{3}^n$. If $n = 1$, then the size of the normal form is also 1.

Case 4. $x = \langle X_1, \ldots, X_l \rangle$ where X_i's are or-sets of a type that does not contain or-sets. Then $normalize(x) = or\_\mu(x)$. Since the or_μ does not change the size of an object, $\mathsf{size}\, normalize(x) < \frac{n}{2}\sqrt[3]{3}^n$ for all $n \geq 2$. If $n = 1$, then $\mathsf{size}\, normalize(x) = 1$.

To complete the inductive proof, we show that after each step of normalization that produces a normalized subobject x'', that is, $x'' = normalize(x')$ for a subobject x' of x, either $\mathsf{size}\, x'' \leq \frac{n}{2}\sqrt[3]{3}^n$ is satisfied if $n = \mathsf{size}\, x' > 1$, or $\mathsf{size}\, x'' = 1$ if $n = 1$. This will complete the proof. Two cases corresponding to application of or_ρ_1 or or_ρ_2 are similar to the case of α, so we show here only the case of application of α.

Let $x = \{x_1, \ldots, x_k\}$ where each x_i is an unnormalized object. Let $x'_i = normalize(x_i)$ and k_i be the cardinality of x'_i, i.e. $k_i = m(x_i)$. Let $n_i = \mathsf{size}\, x_i$. By Theorem 6.2, $k_i \leq \sqrt[3]{3}^{n_i}$. First consider the case when all $n_i > 1$.

Let $x'_i = \langle y^i_1, \ldots, y^i_{k_i} \rangle$, $i = 1, \ldots, k$. By s^i_j we denote $\mathsf{size}\, y^i_j$. By the induction hypothesis,

$$\forall i = 1, \ldots, k: \quad \sum_{j=1}^{k_i} s^i_j \leq \frac{n_i}{2}\sqrt[3]{3}^{n_i}$$

$normalize(x)$ is obtained by applying $\alpha_{\mathbf{d}}$ to $\{\!|x'_1, \ldots, x'_k|\!\}$ and then removing duplicates, i.e. its elements are sets of representatives of $x'_1, \ldots, x'_k$. Since we are interested in an upper bound, we may assume that all the elements of $x'_1, \ldots, x'_k$ are distinct. Then each element of x'_i will be present in $k^{(i)} = (\prod_{j=1}^{k} k_j)/k_i$ sets. Therefore, the upper bound for $\mathsf{size}\, normalize(x)$ can be calculated as the sum of the sizes of all elements of $x'_1, \ldots, x'_k$ multiplied by the number of their occurrences in the normalized object, i.e.

$$\begin{aligned} \mathsf{size}\, normalize(x) &\leq \sum_{i=1}^{k}\sum_{j=1}^{k_i} k^{(i)} s^i_j = \sum_{i=1}^{k} k^{(i)} \sum_{j=1}^{k_i} s^i_j \leq \\ \sum_{i=1}^{k} \frac{n_i}{2} k^{(i)} \sqrt[3]{3}^{n_i} &\leq \sqrt[3]{3}^{n_1 + \ldots + n_i} \sum_{i=1}^{k} \frac{n_i}{2} = \frac{n}{2}\sqrt[3]{3}^n \end{aligned}$$

If all $n_i = 1$, then $\mathsf{size}\, normalize(x) = k = n$. If $n > 1$, then $n \leq \frac{n}{2}\sqrt[3]{3}^n$ and if $n = 1$, that is, $\mathsf{size}\, x = 1$, then $\mathsf{size}\, normalize(x) = 1$.

Now consider the general case, i.e. $n_1, \ldots, n_p > 1$ and $n_{p+1}, \ldots, n_k = 1$. Normalization of x_i for $i > p$ results in a size one object. Let $\sigma_0 = n_1 + \ldots + n_p$ and $\sigma_1 = k - p$. Clearly $\sigma_0 + \sigma_1 = n$. Had we applied $\alpha_{\mathbf{d}}$ only to $\{\!|x'_1, \ldots, x'_p|\!\}$, it would have resulted in an object whose size is bounded above by $\frac{\sigma_0}{2}\sqrt[3]{3}^{\sigma_0}$ according to the calculations for the case where all $n_i > 1$. But taking into account σ_1 size one objects adds size σ_1 to every element of the or-set *normalize*(x). Since there are at most $\sqrt[3]{3}^{\sigma_0}$ such sets, we obtain

$$\mathsf{size}\, \mathit{normalize}(x) \leq \frac{\sigma_0}{2}\sqrt[3]{3}^{\sigma_0} + \sigma_1\sqrt[3]{3}^{\sigma_0}$$

Since $\sigma_0 > 1$, we obtain $\sigma_0 + 2\sigma_1 \leq (\sigma_0 + \sigma_1)\sqrt[3]{3}^{\sigma_1}$. Therefore,

$$\mathsf{size}\, \mathit{normalize}(x) \leq \frac{\sigma_0}{2}\sqrt[3]{3}^{\sigma_0} + \sigma_1\sqrt[3]{3}^{\sigma_0} \leq \frac{n}{2}\sqrt[3]{3}^{n}$$

Finally, if *or*_μ is applied in the process of normalization, it does not change size. Assume $x = \langle x_1, \ldots, x_k\rangle$ where each x_i is an unnormalized object. Let $x'_i = \mathit{normalize}(x_i)$ and $n_i = \mathsf{size}\, x_i$. Assume $n_1, \ldots, n_p > 1$ and $n_{p+1} = \ldots = n_k = 1$. Define σ_0 and σ_1 as in the case of applying α. Then, by the induction hypothesis,

$$\mathsf{size}\, \mathit{normalize}(x) \leq \sum_{i=1}^{p} \frac{n_i}{2}\sqrt[3]{3}^{n_i} + \sigma_1 \leq \frac{\sigma_0}{2}\sqrt[3]{3}^{\sigma_0} + \sigma_1 \leq \frac{n}{2}\sqrt[3]{3}^{n}$$

If all $n_i = 1$, then two cases arise. If $n > 1$, then $\mathsf{size}\, \mathit{normalize}(x) = n \leq \frac{n}{2}\sqrt[3]{3}^{n}$, and if $n = 1$, then $\mathsf{size}\, \mathit{normalize}(x) = n = 1$. This proves the theorem. □

Corollary 6.4 *Let $x = \mathit{normalize}(y)$ and $\mathsf{size}\, x = n$. Then $O(\log n) \leq \mathsf{size}\, y \leq n$.*

The upper bound of Theorem 6.3 is not tight. The following result exhibits a tight upper bound for a large class of objects. This shows that the previous theorem cannot be significantly improved.

Theorem 6.5 *Let x be an object with $\mathsf{size}\, x = n$ containing or-sets. Assume that every subobject of type $\{\langle t'\rangle\}$ has size at least 21, every subobject of type $t' \times \langle t''\rangle$ or $\langle t''\rangle \times t'$ has size at least 6 and every subobject of type $\langle\langle t'\rangle\rangle$ has size at least 3, where t' and t'' do not use the or-set type constructor. Then*

$$\mathsf{size}\, \mathit{normalize}(x) \leq \frac{n}{3}\sqrt[3]{3}^{n}$$

Moreover, for any n divisible by 3 there exists an object x such that $\mathsf{size}\, x = n$ and $\mathsf{size}\, \mathit{normalize}(x) = \frac{n}{3}\sqrt[3]{3}^{n}$.

Proof. We have to rework the base cases only. Since no subobject involving or-sets can have size one, the induction step easily goes through, as in the proof of Theorem 6.3.

The case of applying α was already proved; see the proof of Theorem 6.3. For the case of applying or_ρ_1 or or_ρ_2, we established an upper bound $2n-2$. It is easily seen that $2n-2 \leq \frac{n}{3}\sqrt[3]{3}^n$ for $n \geq 6$. Finally, applying or_μ does not affect the size, and $n \leq \frac{n}{3}\sqrt[3]{3}^n$ for $n \geq 3$.

To show tightness, consider the example from the proof of Theorem 6.3. Let

$$x = \{\langle b_1, b_2, b_3\rangle, \langle b_4, b_5, b_6\rangle, \ldots, \langle b_{n-2}, b_{n-1}, b_n\rangle\}$$

where all b_i's are distinct elements of a base type. Then $\alpha(x)$ contains $\sqrt[3]{3}^n$ elements, each having cardinality $\frac{n}{3}$. Thus, $\mathsf{size}\, \mathit{normalize}(x) = \frac{n}{3}\sqrt[3]{3}^n$. □

The importance of existential queries was emphasized in [17, 18]. Essentially, an existential query asks whether there exists a possibility — in the normal form — satisfying a given property. In terms of $or\text{-}\mathcal{NRA}^+$, if $\mathit{nf}(s) = \langle t\rangle$ and $p : t \to \mathit{bool}$ is a predicate, $\exists(p) : \langle t\rangle \to \mathit{bool}$ is a predicate which is true of $y : \langle t\rangle$ if $or\_map(p)(y) : \langle \mathit{bool}\rangle$ is an or-set containing the true value. Given an object x of type s, one may ask a query $\exists(p)(\mathit{normalize}(x))$. If p is a polynomial time query, then $\exists(p)(\mathit{normalize}(x))$ can be answered in time polynomial in the size of $\mathit{normalize}(x)$. But can it be answered in time polynomial in the size of x?

The following example gives a negative answer to this question, provided $\mathcal{P} \neq \mathcal{NP}$. Assume that $p : \{t \times s\} \to \mathit{bool}$ checks if its input R satisfies the functional dependency $\#1 \to \#2$. That is, if $(x, y) \in R$ and $(x, y') \in R$ imply $y = y'$. This query can be easily implemented in relational algebra and hence in $or\text{-}\mathcal{NRA}$. Now we encode conjunctive normal form Boolean formulae in $or\text{-}\mathcal{NRA}$ as follows. Assume that literals are encoded by elements of a base type b. Each positive literal u is then a pair $(u, \mathit{true}) : b \times \mathit{bool}$ and each negative literal $\overline{u}$ is a pair $(u, \mathit{false}) : b \times \mathit{bool}$. Each disjunction of formulae is encoded as an or-set of encodings of its components, and each conjunction of formulae is encoded as a set of encodings of its components. Now let $x_\psi : \{\langle b \times \mathit{bool}\rangle\}$ be an encoding of a Boolean formula ψ. Then $\exists(p)(\mathit{normalize}(x_\psi))$ evaluates to true if and only if ψ is satisfiable. Thus, we cannot evaluate existential queries on normal forms in polynomial time under the assumption that $\mathcal{P} \neq \mathcal{NP}$.

7 Conclusion and Future Work

In this paper we considered a simple semantic model that cleanly integrates sets and or-sets, taking into account their intended meaning. We showed that there are two levels for manipulating sets and or-sets — structural and conceptual — and we extended proposals of [3, 5, 33] to formulate a query language for the structural manipulation of or-sets. We defined the concept of normalization of objects involving or-sets and proved its coherence. That allowed us to include normalization as a primitive to obtain the language for manipulation of or-sets at the conceptual level. We proved that normalization is lossless and established upper bounds on the cost of normalization.

The language *or*-$\mathcal{NRA}^+$ has been implemented on top of Standard ML. The implementation provides an interface which includes the operations of *or*-$\mathcal{NRA}^+$ and a few additional operations that elevate *or*-$\mathcal{NRA}^+$ to capture the power of a nested bag language [25]. The package also includes additional features such as creation and destruction of objects, structural recursion on sets and or-sets, input and output facilities. It also comes equipped with several libraries of derived functions. For example, the library of set functions includes membership test, set difference, inclusion test, cartesian product, etc., and their analogs for or-sets which, as follows from the results of [5] and this paper, are definable in *or*-$\mathcal{NRA}^+$. Another library defines a lifting of linear orders from base types to arbitrary types which is definable in *or*-$\mathcal{NRA}$ as demonstrated in [26]. A complete description of this implementation of *or*-$\mathcal{NRA}^+$ can be found in [12].

There are many further problems which we would like to investigate. The use of bags in the proof of the coherence theorem suggests adding or-sets to a bag language. We would like to study the language obtained by combining the or-set component of *or*-$\mathcal{NRA}$ and the standard nested bag language such as the one in [25]. Such a language gives rise to interesting equational theories which can lead to useful optimizations. In addition to the monad equations of [5], every diagram in the proof of Theorem 4.2 gives rise to a new equation.

We have seen that normalization can be quite expensive. Therefore, there is a need for techniques that make query evaluation faster. Using functional style language and its implementation on top of ML suggests the use of lazy evaluation for possible optimization of some queries. For instance, implement normalization in such a way that elements of a normal form are produced as elements of a stream. Then, if an existential query is evaluated over a normal form, elements of the normal form are produced as they are needed, and if the test is satisfied, the evaluation stops without producing the whole normal form. Such a mechanism for query evaluation has recently been developed by one of the authors [23].

Yet another idea is the complexity-tailored design of Imielinski, van der Meyden, and Vadaparty [16] when queries are forced to run in polynomial time by, for instance, obtaining additional information about some of the or-sets, thus reducing the size of the normal form. In [16] a logical language was used. We would like to see if the idea can be worked out for our languages.

There are various sophisticated order-theoretic models of partial information in databases — sandwiches [6], mixes [10], snacks [31, 30], and their generalizations [31, 22]. They are used when a real world situation can be approximated from below and above by information in a database. These structures enjoy universality properties and therefore can be incorporated into the programming language syntax [22]. We have recently shown [22] that the intimate connection between or-sets and the Smyth powerdomain can help us use or-sets for a suitable representation of those approximation models in the context of database programming languages like *or-$\mathcal{NRA}$*. We plan to further investigate the applicability of such models to the study of or-sets.

Our languages have been extended to include variant types. It is known that the coherence result still holds in the extended languages. The validity of the remaining results of this report remains to be checked for this extension.

Acknowledgements. The authors are grateful to Tim Griffin, Elsa Gunter, Anthony Kosky, Shamim Naqvi, Val Tannen and especially Peter Buneman for many interesting discussions. Most of this work was done when both authors were at the University of Pennsylvania. We gratefully acknowledge the support of an AT&T Doctoral Fellowship and NSF Grant IRI-90-04137 (for Libkin) and NSF Grant IRI-90-04137 and ARO Grant DAALO3-89-C-0031-PRIME (for Wong).

References

[1] S. Abiteboul, C. Beeri, On the power of languages for the manipulation of complex objects, *in* "Proceedings of International Workshop on Theory and Applications of Nested Relations and Complex Objects," Darmstadt, 1988.

[2] S. Abiteboul, P. Fischer, and H.-J. Schek, editors, "LNCS 361: Nested Relations and Complex Objects in Databases," Springer-Verlag, 1989.

[3] V. Breazu-Tannen, P. Buneman, S. Naqvi, Structural recursion as a query language, *in* "Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece, August 1991," 9–19.

[4] V. Breazu-Tannen, R. Subrahmanyam, Logical and computational aspects of programming with sets/bags/lists, *in* "LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991," 60–75.

[5] P. Buneman, S. Naqvi, V. Tannen, and L. Wong, Principles of programming with complex objects and collection types, *Theoretical Computer Science*, to appear. Extended abstract *in* "LNCS 646: Proceedings of International Conference on Database Theory," Berlin, Germany, October 1992.

[6] P. Buneman, S. Davidson, A. Watters, A semantics for complex objects and approximate answers, *Journal of Computer and System Sciences* **43** (1991), 170–218.

[7] P. Buneman, A. Ohori, A. Jung, Using powerdomains to generalize relational databases, *Theoretical Computer Science* **91** (1991), 23–55.

[8] N. Dershowitz and J.-P. Jouannand, Rewrite systems, *in* "Handbook of Theoretical Computer Science", North Holland, 1990, 243–320.

[9] K.E. Flannery and J.J. Martin, Hoare and Smyth power domain constructors commute under composition, *Journal of Computer and System Sciences* **40** (1990), 125–135.

[10] C. Gunter, The mixed powerdomain, *Theoretical Computer Science* **103** (1992), 311–334.

[11] C. Gunter and D. Scott, Semantic Domains, *in* "Handbook of Theoretical Computer Science", North Holland, 1990, 633–674.

[12] E. Gunter and L. Libkin, OR-SML: A functional database programming language for disjunctive information and its applications, *in* "LNCS 856: Proceedings of Database and Expert Systems Applications, Athens, September 1994," 641–650.

[13] R. Harper, R. Milner, and M. Tofte, "The Definition of Standard ML," MIT Press, 1990.

[14] R. Heckmann, Lower and upper power domain constructions commute on all cpos, *Information Processing Letters* **40** (1991), 7-11.

[15] T. Imielinski and W. Lipski, Incomplete information in relational databases, *Journal of the ACM* **31** (1984), 761–791.

[16] T. Imielinski, R. van der Meyden, and K. Vadaparty, Complexity tailored design: A new design methodology for databases with incomplete information, *Journal of Computer and System Sciences*, to appear.

[17] T. Imielinski, S. Naqvi, and K. Vadaparty, Incomplete objects — a data model for design and planning applications, *in* "Proceedings of ACM-SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991," 288–297.

[18] T. Imielinski, S. Naqvi, and K. Vadaparty, Querying design and planning databases, *in* "LNCS 566: Proceedings of 2nd International Conference on Deductive and Object-Oriented Databases, Munich, Germany, December 1991," 524–545.

[19] L. Libkin, A relational algebra for complex objects based on partial information, *in* "LNCS 495: Proceedings of Symposium on Mathematical Fundamentals of Database Systems, Rostock, May 1991," 36–41.

[20] L. Libkin, An elementary proof that upper and lower powerdomain constructions commute, *Bulletin of the EATCS* **48** (1992), 175-177.

[21] L. Libkin, "Aspects of Partial Information in Databases," PhD Thesis, University of Pennsylvania, August 1994. Available from
`http://www.cis.upenn.edu/~libkin/home.html`.

[22] L. Libkin, Approximation in databases, *in* "LNCS 893: Proceedings of 5th International Conference on Database Theory, Prague, January 1995," 411–424.

[23] L. Libkin, Normalizing incomplete databases, *in* "Proceedings of 14th ACM Symposium on Principles of Database Systems, San Jose, California, May 1995," 219–230.

[24] L. Libkin, L. Wong, Semantic representations and query languages for or-sets, *in* "Proceedings of 12th ACM Symposium on Principles of Database Systems, Washington, D. C., May 1993," 37–48.

[25] L. Libkin, L. Wong, Some properties of query languages for bags, *in* "Proceedings of 4th International Workshop on Database Programming Languages, Manhattan, New York, August 1993," 97–114.

[26] L. Libkin, L. Wong, Conservativity of nested relational calculi with internal generic functions, *Information Processing Letters* **49** (1994), 273–280.

[27] E. Moggi, Notions of computation and monads, *Information and Computation* **93** (1991), 55–92.

[28] J. Moon and L. Moser, On cliques in graphs, *Israel Journal of Mathematics* **3** (1965), 23–28.

[29] S. Naqvi and S. Tsur, "A Logical Language for Data and Knowledge Bases," Computer Science Press, 1989.

[30] T.-H. Ngair, "Convex Spaces as an Order-theoretic Basis for Problem Solving," PhD Thesis, University of Pennsylvania, August 1992.

[31] H. Puhlmann, The snack powerdomain for database semantics, *in* "LNCS 711: Mathematical Foundations of Computer Science, Gdansk, September 1993," 650–659.

[32] B. Rounds, Situation-theoretic aspects of databases, *in* "CSLI 26: Proceedings of 1991 Conference on Situation Theory and Applications," 229-256.

[33] P. Wadler, Comprehending monads, *Mathematical Structures in Computer Science* **2** (1992), 461–493.

[34] G. Winskel, Powerdomains and modality, *Theoretical Computer Science* **36** (1985), 127-137.