WINDOW FUNCTIONS

by
D. Maier, D. Rozenshtein, & D. S. Warren

Oregon Graduate Center

OGC Technical Report No. CS/E 84-002

WINDOW FUNCTIONS

David Maier† *
Oregon Graduate Center

David Rozenshtein†
Rutgers, the State University of New Jersey

David S. Warren†
State University of New York at Stony Brook

SUMMARY

We discuss the philosophy, history and theory of window functions. Window functions are a means to treat a relational database as a semantic whole, rather than as an arbitrary collection of relations. Simply stated, a window function maps a database state and a relation scheme to a relation over the scheme. Window functions are the basis for all existing universal scheme interfaces. We present an assumption inherent in such interfaces, the *unique role assumption*.

Window functions have evolved along two paths, giving rise to computational definitions and weak instance definitions. We examine several examples of each type of window function, with special attention to the association-object window function of PIQUE. We then look at properties we feel a reasonable window function should satisfy, notably the *containment condition* and *faithfulness*. We also define *implicit objects*, which are relation schemes that a window function treats in a special manner, and are useful for describing the behavior of window functions.

1. The Why and What of Window Functions

As Maier, Vardi and Ullman <MUV> note, the relational data model has gone far towards *physical* data independence, but has not achieved the goal of *logical* data independence. That is, users of relational systems are relieved of specifying access paths within the structure of a single relation, but they still must navigate between relations. Users and application programs are protected from changes in the physical implementation of relations, but not from changes in the logical structure of a database, such as decompositions made for normalization or efficiency reasons.

Universal scheme interfaces are an attempt at logical data independence. In a universal scheme interface, all the semantics of the database is loaded onto

†Work supported by NSF grant IST 81-04834.
*Work supported by NSF grant MCS 82-07216.

the attributes. Queries are phrased in terms of attributes; the user need not know which attributes are in which relations. The hope is that the attribute names correspond naturally to entities in the real world, and that the users have an intuitive semantics for these entities and their relationships that is close to the actual semantics of the database. In a universal scheme interface, a database is presented as a semantic whole, accessible through its attributes alone. In the sequel, U will denote the universe of attributes in a database: the *universal scheme*.

There are several universal scheme systems extant and under development. The first were APPLE <CK> and that of Shenk and Pinkert <SP>. More recent systems are q <AK>, System/U <K, KKU, KU, U11>, PIQUE <MRSSW, MW, Ro>, Parafrase <KMRS, KS>, FIDL <Ba>, DURST <BB> and that of Arazi-Gonczarowski <A-G>. Many of the same issues are addressed by Sowa's work on conceptual graphs <So>, and by others' work on automatic navagation in a database <Li, Su, Za>.

Query processing in a universal scheme system can be cast as a two-stage procedure:

1. The set of attributes, call it X, appearing in the query is determined. Then, on the basis of the state of the database, a relation r over scheme X is generated. (If the query contains several variables, the attributes associated with each variable are used to compute separate relations.)

2. Further operations specified by the query are applied to $r(X)$ to generate the answer.

These stages are called *binding* and *evaluation*, respectively.

Example 1: Consider a simple database courses with the relations *taking*(STUDENT COURSE) and *teaching*(FACULTY COURSE). In response to the query

retrieve FACULTY **where** STUDENT = "Andrews"

the PIQUE system will construct a relation r on FACULTY STUDENT. Presumably,

r will be

$\pi_{\text{FACULTY STUDENT}}(taking \bowtie teaching)$.

In the evaluation stage of processing the query, PIQUE applies the selection $\sigma_{\text{STUDENT}=\text{Andrews}}$ to r to get the final answer to the query.

The generation of a relation r on scheme X in the binding step is done, implicitly or explicitly, through the application of a *window function*. A window function f maps a relation scheme $X \subseteq U$ and a database state d to some relation r with scheme X: $f(X, d) = r(X)$. In our development, we will "curry" f and treat it as a functional that maps relation schemes to functions from database states to relations:

f :relation schemes $\rightarrow$ (database states $\rightarrow$ relations).

The functional signature above doesn't tell the whole story. First, we assume a single, fixed database scheme for all the database states. Second, f is polymorphically typed in that $f(X)$ is a function with relations over scheme X as its range.

We use an alternative notation for window functions in the sequel. A window function will be denoted by brackets, possibly subscripted: $[\cdot]$, $[\cdot]_I$, $[\cdot]_{LJ}$. Application of a window function to a scheme X is denoted $[X]$ and called the *window on* X for that window function. The application of the window on X to database state d is $[X](d)$; if d is understood, we sometimes use simply $[X]$.

While the stages of binding and evaluation do interact, they are loosely-coupled. Changes to the window functions can be made without changing the procedures in the evaluation step, although such changes will mean different answers for queries. All the universal scheme systems mentioned above seem to conform to the two-step paradigm. It is interesting to note that they do not vary widely in the expressive power for operations to be applied in the second step (though their query languages do differ in syntax). The significant differences between the systems are the window functions used.

A logical question at this point is why the same effect as window functions cannot be captured with virtual relation or view mechanisms, as in conventional relational systems? The first answer is that the use of virtual relations, while it hides the computations needed to produce them, still requires the user to know the names of all the virtual relations and their schemes. The second answer is that window functions *are* a view definition mechanism. However, defining a window function on a specific database need not require explicit definition of a separate window for each set of attributes. Rather, in most universal scheme systems there is a uniform discipline for deriving a windows from semantic information about the database, such as the database scheme, and functional and join dependencies. Unlike an arbitrary set of virtual relations, the windows in a universal scheme system are meant to display some manner of semantic consistency. The term *window* conveys the image of a consonant set of views into a single database world.

For a universal scheme interface to a database to be practicable, the database must at least satisfy the *universal relation scheme assumption* (URSA). URSA states that any attribute in U corresponds to the same class of entities wherever it appears. NUMBER cannot refer to serial numbers of equipment in one place and social security numbers in another; otherwise, there is no way to distinguish one class of entities from the other in the query language. URSA requires that, at a minimum, an attribute have the same domain wherever it appears. For relational systems in which the available domains are just integer, real and string, this requirement does not mean much. It is more a constraint on the conceptual model, where each class of entities is represented by a separate domain.

Positing the existence of window functions makes an assumption stronger than URSA, which we call the *unique role assumption* (URA). URA requires that an attribute not only always represent the same class of entities, but also always represent the same role for that class. DATE may not represent dates both in the role of birthdate and the role of hiring date. Put another way, URA means "the scheme determines the connection": For any set of attributes, there is at most one connection among them. In particular, no two database relations have the same scheme. Without URA, a universal scheme system has no way to tell which relationship among a set of entities is intended when those entities are mentioned together. If DATE played two roles, as above, there is no way to tell if the window [EMPLOYEE DATE] is asking for the connection between employees and birthdates or between employees and hiring dates.

It is unlikely that a relational database designed without URA in mind would satisfy that assumption. To get satisfaction, some attributes will probably have to be renamed, in order to distinguish the roles portrayed. The two roles of DATE above could be distinguished as BIRTHDATE and HIRING_DATE. Such renaming can produce problems. It may not be apparent, after renaming, that different attributes represent the same class of entities, and the proliferation of attribute names can become unwieldy. We are currently considering ways to handle such problems by explicitly incorporating a generalization hierarchy as part of the database description <MRS, Ro>. Of course, many others have looked at generalization in relational databases <BK, Sc2, SS>.

Not all the universal scheme systems mentioned previously strictly enforce URA. Carlson and Kaplan, in their APPLE system <CK>, try to define window functions upon databases that do not necessarily satisfy URA, or even URSA. They search for a series of natural joins and equijoins that will connect two sets of attributes, guided by functional dependencies (FDs). Their method does not construct windows for arbitrary relation schemes. Rather, it concentrates on connecting pairs of single attributes. Trying to impose a universal scheme view after the fact upon a database that does not satisfy URA causes several complications. They must maintain explicit information on comparability of attributes. Their method for computing expressions for windows can generate multiple connections between a pair of attributes, actually giving several windows for the same relation scheme. They discuss several ways to ameliorate the problem, but, ultimately, the user must select among connections when several exist. Because of such ambiguities, APPLE is not quite a full-fledged universal scheme interface.

The query system q <AK> does not make any assumptions about the database. There is a "relfile" containing a list of schemes for stored and virtual relations, and procedures for computing virtual relations. To generate $[X]$, q sequentially scans the relfile for the first scheme of a stored or virtual relation containing X. A computation is performed, if necessary, and the corresponding relation is projected onto X. Nothing constrains the type of computations allowed to generate virtual relations; they need not even use the stored relations of the database. While q's mechanism produces a single window for any relation scheme, the views these windows present are not necessarily consistent with each other or with the database. Thus, URA is satisfied for all windows derived from the same virtual relation, but not necessarily for the database as a whole. In practice, the computations used to derive virtual relations usually consist entirely of joins, and the database does satisfy URA. One other problem with q is that the expressions for virtual relations must be given explicitly. Work is currently underway on methods to generate those expressions from dependency information about the database <Ho3>.

We do not construe URA so strongly as to prohibit multiple semantic connections among a set of attributes. We only intend that the system takes one of those connections as the most natural, and will make that connection automatically. Other connections must be made explicitly by the user.

2. Types of Window Functions

Window functions are a particular approach to the general problem of inference and deduction from a knowledge base. Here we are working with highly-structured data, within a limited domain of discourse, and with certain simplifying assumptions. We hope thereby to get more determinism and efficiency than from a more general knowledge-based deduction system.

The window functions used in the universal scheme systems mentioned, and in various theoretical studies to be discussed, are not always expressed in the form given here. Often the definition of a window function is implicit within some computational method. In particular, some systems compute $[X](d)$ directly, never realizing an explicit expression for $[X]$. In other studies, the term *connection* is used variously for window function and window. The bracket notation used here follows the "output functions" of Maier <Ma1>.

There are two main concerns in defining a window function. One is that it have a reasonable semantics, the other is that it be efficient to compute. Most window functions can be placed along one of two lines of development, corresponding to which concern is emphasized. There are window functions based on straight computational definitions, where the semantic assumptions may be fairly rigid. Other window functions are based on weak instances, and place more importance on the semantics. Weak instances were originally introduced to study another aspect of databases as semantic wholes: global satisfaction of dependencies. We shall see in the next section how computational definitions evolved to have more semantic content. In the following section, we shall see the development of more efficient computation methods for weak instance windows.

In the rest of this paper, we shall let **R** be the database scheme $\{R_1, R_2, \ldots, R_p\}$, and let $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ be a database over **R**. Thus, U is the union of the schemes in **R**.

2.1. Computational Window Function Definitions

Most initial work on computational definitions for window functions made an assumption stronger than URA. That assumption is the *universal instance assumption* (UIA), which states that the database relations are all projections of a single relation I over U. (I is called a *universal instance*.) That is,

$$r_i = \pi_{R_i}(I),\ 1 \le i \le p.$$

UIA is a requirement for "universal extension," where URA only requires "universal intention."

Under UIA, the window $[X]$ is defined as $\pi_X(I)$. We denote this window function as $[\cdot]_I$. The presumption is that I can be recovered from d as $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_p$, or at least $\pi_X(I)$ can be recovered, for certain X. A number of groups have studied the question of when all or part of I can be recovered from its projections <AC, BMSU, MMSU, Ri>. In short, the question they address is whether the dependencies that I must satisfy imply that $r_1, r_2, \ldots, r_p$ have a lossless join.

The work on UIA-based window functions concentrates on finding, for a given X, an alternative expression E such that $\pi_X(E) = \pi_X(I)$. The hope is that E can be computed more efficiently than the join of all the relations. Shenk and Pinkert <SP> were among the first to look at this question. They concentrated on *lossless subjoins*: a subset $\{s_1, s_2, \ldots, s_m\}$ of d such that $\pi_X(s_1 \bowtie s_2 \bowtie \cdots \bowtie s_m) = \pi_X(I)$. In their study, the only dependencies considered are FDs arising from keys, and the only joins are lossless joins of pairs of relations. The join of $r(R)$ and $s(S)$ may only be taken when $R \cap S \rightarrow R$ or $R \cap S \rightarrow S$. This restriction guarantees small intermediate results, as $|r \bowtie s|$

will be no larger than max($|r|$, $|s|$). They attempt to find a lossless subjoin with the fewest relations by a dynamic programming method. Their method permits a given relation to appear more than once in a join expression.

Lozinskii <Lo> also uses only key FDs and pairwise lossless joins. However, he is looking for lossless joins with a single *source* relation. The join expression starts off with the source relation, and all subsequent relations are joined with it, so there is only a single intermediate result. That is, the expression tree for the join expression is actually a "vine." For a given X, he is interested in finding all the lossless subjoins for computing $[X]$, so that he may pick the best according to a given cost function. Honeyman <Ho1> is also looking for a sequence of joins emanating from a single source, but he uses an arbitrary set of FDs (as long as they are embedded in the database scheme), and allows projections of relations into the join expression. (He terms such joins *extension joins*.)

UIA is well known to have its shortcomings. It is a hard condition to test in general, and it is not realistic in many applications. The usefulness of UIA-based window functions is probably limited to a database that was originally a single relation, but was later decomposed to remove redundancy. To avoid these computational and semantic restrictions, several researchers have defined window functions where UIA is used to determine the lossless subjoins, but it is not expected to actually hold for an arbitrary database state.

Osborn <Os> defined $[X]$ as the union of X-projections of all lossless subjoins covering X. Her join expressions also start with a source relation that is augmented through pairwise lossless joins. The semantic assumption is that all connections corresponding to lossless joins are equally meaningful. Her dependency information is solely key FDs. We denote her window function by $[\cdot]_{LJ}$.

The designers of System/U <K, KKU, KU, Ul1>, and Fagin, Mendelzon and Ullman <FMU> advocate a UIA approach, with $[X]$ being

$\pi_X(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p)$.

In System/U, to avoid problems when relations may not join completely, the join is minimized under weak equivalence. That is, relations are pruned from the expression above under the assumption that the relations do join completely.

Example 2: Consider a database on attributes A (advisor), S (student), C (course) and I (instructor), with relations $r(\text{A S})$, $r(\text{S C})$, $r(\text{C I})$. Minimization under weak equivalence prunes the expression

$$E_1 = \pi_{\text{SI}}(r(\text{A S}) \bowtie r(\text{S C}) \bowtie r(\text{C I}))$$

down to

$$E_2 = \pi_{\text{SI}}(r(\text{S C}) \bowtie r(\text{C I})).$$

E_1 and E_2 are equivalent on databases where every student has an advisor, but E_2 can have more tuples when evaluated on a database where this is not the case.

Minimization under weak equivalence is not done just as a computational optimization, but also for its semantic overtones. In the last example, the connection of a student to an instructor via a course shouldn't be influenced by the presence or absence of an advisor for the student. The technique used for this transformation is tableau minimization <ASU1, ASU2>, which has the advantage that dependency information can be brought in to help reduce the number of joins. We shall let $TM(E)$ denote the tableau minimization of an expression E, and let $[\cdot]_{SU}$ represent the System/U window function, so

$$[X]_{SU} = TM(\pi_X(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p).$$

One problem with this approach is that there can be several alternatives for $TM(E)$. Those alternatives are equivalent if UIA holds, but not for arbitrary states of the database. In cases where TM yields multiple expressions equivalent to the original, the union of these expressions is used.

Example 3: Consider a database with relations $r(A\ C\ D)$, $r(B\ C\ D)$, and $r(D\ E)$. (Since schemes are unique in a URA database, we may call all relations r.) For the window $[C\ E]_{SU}$, the expression can be minimized to $\pi_{CE}(r(A\ C\ D) \bowtie r(D\ E))$ or $\pi_{CE}(r(B\ C\ D) \bowtie r(D\ E))$. The union of these two expressions is used to compute $[C\ E]_{SU}$

The System/U approach gives a semantically reasonable window function when the the database scheme **R** is *acyclic*. (See Beeri, et al. <BFMMUY, BFMY> for material on acyclic database schemes.)
and Ullman <MU> note that in the presence of cycles, the definition above for $[\cdot]_{SU}$ may not represent a natural connection, or is likely not to be the particular connection a user had in mind.

Example 4: Consider a database on attributes B (bank), L (loan), A (account), and C (customer), with relations r(B L), r(L C), r(B A), and r(A C). Notice the cycle the relation schemes form. The database contains information about loans and accounts at banks, and about which customers took out the loans and own the accounts. In System/U, $[\mathrm{B\ C}]_{SU}$ will be computed as

$$\pi_{\mathrm{B\,C}}(r(\mathrm{B\ L}) \bowtie r(\mathrm{L\ C}) \bowtie r(\mathrm{B\ A}) \bowtie r(\mathrm{A\ C})).$$

This expression gives all customers who have both a loan and an account at a bank, which is not a very natural meaning for the bank-customer connection.

Using the ideas of Sciore <Sc1>, and Ullman define *maximal objects* to break up cycles in the database scheme. A maximal object is a subset of U. Let **M** be a set of maximal objects. To compute $[X]$, for each $W \in \mathbf{M}$ such that $X \subseteq W$, they form the join of all relations whose schemes are contained in W. The join for each applicable maximal object is projected onto X and then pruned by tableau minimization. The window $[X]$ is finally obtained as the union of all these expressions. We denote this window function by $[\cdot]_{\mathbf{M}}$, for a set **M** of maximal objects.

Example 5: Returning to the last example, let **M** = {B L C, B A C}. This set of maximum objects says there are two ways to connect bank and customers, through loans and through accounts, where, without maximal objects, there was only a single connection, through loans and accounts simultaneously. Here, $[\text{B C}]_{\mathbf{M}}$ =

$$\pi_{\text{B C}}(r(\text{B L}) \bowtie r(\text{L C})) \cup \pi_{\text{B CD}}(r(\text{B A}) \bowtie r(\text{A C})).$$

Maximal objects augment a database scheme with semantic information about which connections are most meaningful.
and Ullman also present several methods of automatically generating a set of maximal objects for a given database scheme, using functional and join dependencies. They allow that a set of maximal objects so generated might be further tailored by the database designer. They also permit different users to employ different sets of maximal objects, reflecting different views on what the important connections are. Another capability of maximal objects is that they allow the database designer to indicate that certain attributes are so semantically distant that no connection among them should be derived automatically. The window on Y can be identically the empty relation, if no maximal object in **M** contains Y, such as $[\text{L A}]_{\mathbf{M}}$ in the last example.

Our view in developing window functions for the PIQUE query language is that data dependencies by themselves are not sufficient for inferring the desired connections among attributes. Also, minimization under weak equivalence is complex process, and we doubt many database designers will understand its exact effect on the meaning of a window. There may be several connection semantics that agree with a given set of data dependencies. We have defined window functions based on a set **A** of *associations* and a set **O** of *objects* <MW>.

Associations are sets of attributes that represent permissible units of update. An association is a possible scheme for a tuple entered into the database. We let $r(R)$ denote all the tuples in the database whose scheme is the association R. The reader will not be far from right to assume that the set **A** of associations is the database scheme, although, in practice, heterogeneous tuples may be stored in a single relation through the use of placeholders. We depart from usual practice in that we allow subassociations of associations. We adopted the term association in place of relation scheme to emphasize this departure. We permit tuples over both associations R and S, where R is a proper subset of S. Under UIA, there is not much sense in having relations over both R and S, as $r(R)$ will be $\pi_R(r(S))$. Without UIA, relations over both schemes do make sense, although URA does dictate certain restrictions, as we shall see in Section 3.

Example 6: In an association-object database, we could have both an association C I, meaning a course is taught by an instructor, and a containing association C I S, meaning a course is taught by and instructor to a student.

Objects are also sets of attributes, and represent units of retrieval. Objects dictate which joins will be used to construct window functions. For each object $W \in \mathbf{O}$, we assume that W is the union of associations in **A**, and define a relation on W, denoted $r'(W)$, by joining on all associations contained in W:

$$r'(W) = \bowtie_{R \in \mathbf{A}, R \subseteq W} r(R).$$

We then define window function $[\cdot]_{\mathbf{AO}}$ from these object relations by projecting the appropriate object relations.

$$[X]_{\mathbf{AO}} = \bigcup_{W \in \mathbf{O}, W \supseteq X} \pi_X(r'(W)).$$

Example 7: Consider a database where $\mathbf{A} = \{S\ C,\ C\ I\}$, meaning a student takes a course and a instructor teaches a course. Let $\mathbf{O} = \{S\ C\ I\}$. For this choice of **A** and **O**, the connection between student and instructor is by joining on course. If we add S C I as an association, then we explicitly store from which instructor a student is taking a course. Now, the connection between student and instructor will be the projection of r(S C I), rather than students taking a course and a instructor teaching some section of that course.

We shall argue in Section 3 for the desirability of the set **O** being closed under nonempty intersection.

2.2. Weak Instance Window Functions

The second main path along which window functions have evolved is weak instance definitions. Weak instances, and their cousins, representative instances, were first introduced as a means for discussing global satisfaction of a set of dependencies by a database <Ho, Gr, Va>, and for inferring missing information in a database state <Ma1, Wa>. They have also been used recently to study the equivalence of database schemes <Me>. We show now how weak instances are used to define window functions.

A relation $I(U)$ is a *containing instance* for database d if

$$\pi_{R_i}(I) \supseteq r_i,\ 1 \le i \le p.$$

For a set of dependencies **C**, $I(U)$ is a *weak instance under* **C** for d if $I \in SAT(\mathbf{C})$ and I is a containing instance of d. (I satisfies all the constraints in **C**.) We abbreviate "weak instance under **C**" to **C**-WI. A database state need not always have a weak instance.

Example 6: The database

r(

A	B
1	2

) r(

B	C
2	3

) r(

A	C
4	5

)

has weak instances under $\mathbf{C} = \{B \rightarrow C\}$. One is

A	B	C
1	2	3
4	5	6

If <4 5> in $r(A\ C)$ is changed to <1 5>, then the database has no weak instances under $B \rightarrow C$.

The philosophy of the weak instance approach is that a database state represents partial information about some universal instance. However, since the information is incomplete, it does not completely determine which universal instance a database state d represents. The weak instances of the state are the possible universal instances. We can use weak instances to define a window function. The window on X will be all those X-components of tuples that appear in every weak instance of d:

$$[X](d) = \bigcap_{I \text{ a } \mathbf{C}\text{-WI}} \pi_X(I)$$

Since different choices for **C** give different windows, we distinguish the window function for **C** as $[\cdot]_{\mathbf{C}}$.

Of course, this definition for $[X]_{\mathbf{C}}$ does not give an effective method of computation. If the dependencies in **C** can be used in a chase computation <ABU, MMS>, then *representative instances* can be used to compute $[X]_{\mathbf{C}}$.

Definition: An *extended instance* (EI) T over scheme U is a relation over U that contains both values and marked nulls.

We assume nulls are marked with numbers to distinguish them, and we will refer to the tuples of an EI as rows to avoid confusion. One particular EI of interest is derived from a database state d. First, pad out each relation in d to have scheme U using distinct marked nulls. Next, take the union of the padded relations. We denote the result by T_d. In parts of the sequel, we will need to keep track of the relation from which each row in T_d was generated.

A representative instance for a database d is formed in two stages. First, form T_d. Second, the chase procedure for dependencies in **C** is applied to T_d to equate nulls and generate new rows. The result of the second stage is the representative instance for d under **C**, which we denote $RI_{\mathbf{C}}(d)$. To summarize:

$$RI_{\mathbf{C}}(d) = chase_{\mathbf{C}}(T_d).$$

It is possible that a contradiction to an equality-generating dependency is encountered during the chase, in which case we define the representative instance to be the empty relation on U.

Maier, Ullman and Vardi <MUV> show that for a large class of dependencies

$$[X]_{\mathbf{C}} = \pi_{\downarrow X}(RI_{\mathbf{C}}(d)),$$

where $\pi_{\downarrow X}$ is X-*total projection*: the X-component of all tuples that have no nulls in the X-columns.

Sagiv <Sa1> considered $[X]_F$, where F is a set of FDs expressed by keys. He gave a condition on database states, the *modified foreign key constraint*, that, together with local satisfaction of F, guarantees that a database state will have at least one F-WI. Sagiv later defined the *uniqueness condition* on FDs and database schemes, which ensures that any locally satisfying database state has a weak instance <Sa2>. His uniqueness condition is a characterization of independence[1] for database schemes under key FDs. We shall denote a window function based on representative instances under FDs that satisfy Sagiv's uniqueness condition by $[\cdot]_{\mathbf{K}}$, where $\mathbf{K}$ is the set of key FDs.

Yannakakis <Ya> looked at $[X]_{\mathbf{C}}$ where $\mathbf{C}$ is a single join dependency (JD) corresponding to the database scheme, $\mathbf{R}$. That is, $\mathbf{C} = \{*[R_1, R_2, \cdots, R_p]\}$. We denote such a window function by $[\cdot]_{*\mathbf{R}}$.

Although representative instances give a means to compute $[X]_{\mathbf{C}}$, the method is not very manageable, especially when the database is large and $\mathbf{C}$ contains tuple-generating dependencies. Sagiv <Sa2> showed that $[X]_{\mathbf{K}}$ can be computed as the union of projections of *extension joins*, a particularly efficient type of join <Ho>. Yannakakis <Ya> showed that $[X]_{*\mathbf{R}}$ can be computed efficiently when $\mathbf{R}$ is acyclic. Maier, Ullman and Vardi <MUV> give conditions for $[X]_{\mathbf{C}}$ to be first-order (computable with algebraic operations), although they are not particularly concerned with the efficiency of computation. We give here a

[1]A database scheme $\mathbf{R}$ is *independent* relative to a set $\mathbf{C}$ of dependencies if any database d on $\mathbf{R}$ that locally satisfies $\mathbf{C}$ also globally satisfies $\mathbf{C}$ (that is, has a $\mathbf{C}$-WI).

generalization of Sagiv's result, where the FDs need not be keys, but can be any independent set of FDs embedded in **R**. Our result uses a slightly modified variety of extension joins.

Definition: Let $r(R)$ and $s(S)$ be relations. The join $r \bowtie \pi_{XY}(s)$ is an *FD-join of* r *with* s *on* X→Y if

1. s satisfies $X \rightarrow Y$, and
2. $X \subseteq R$.

Observations: Note that X must be in $R \cap S$. In an extension join, X must equal $R \cap S$. On databases that satisfy UIA, $r \bowtie \pi_{XY}(s) = r \bowtie \pi_{ZY}(s)$, where $Z = R \cap S$. In fact, these joins give the projection of the universal instance onto $R\ Y$. Also notice that the FD-join of r with s on $X \rightarrow Y$ satisfies all the FDs that r satisfies, plus any FD $W \rightarrow Z$ that s satisfies, where $W\ Z \subseteq X\ Y$. We still have the efficiency of extension joins for FD-joins, as an FD-join of r with s will have no more tuples than r does.

Definition: For database $d(\mathbf{R}) = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ and a set of FDs F, we say E is an *FD-join expression on* d *under* F if

1. E is r_i for some $r_i \in d$, or
2. E is $(E' \bowtie \pi_{XY}(r_i))$, where
   $X \rightarrow Y \in F^+$,
   $scheme(E') \supseteq X$,
   $X\ Y \subseteq R_i$, and
   E' is an FD-join expression of d under F.

That is, E represents a sequence of FD-joins involving relations in d. Note that we only needed the r_i's as placeholders, so we will usually write of an FD-join expression on **R**.

Theorem 1: Let F be an independent set of FDs, embedded in database scheme **R**. For any X subset U, there is an expression E that is the union of projections of FD-join expressions on R under F such that $[X]_F(d) = E(d)$ for every database $d(\mathbf{R})$ satisfying F[2].

Proof: This proof will have several definitions and propositions interpolated in it. We intend that the objects mentioned in the theorem statement and this proof carry over into those definitions and propositions.

We first note that $[X]_F = [X]_G$ for any set of FDs G equivalent to F and that the definition of FD-join expression depends only on the closure of F, so we can make some assumptions on the form of F. We can modify F to an equivalent set of FDs as long as the new set can still be embedded in **R**. First, assume that every FD in F is *canonical:* a single attribute on the right side and no extraneous attributes in the left side. Second, we assume the FDs in F are ***locally closed*** under implication: For each R_i, all the canonical FDs in F^+ that apply to R_i are in F. Note that we can modify F to satisfy these two assumptions without affecting embeddability, that there is exactly one set of FDs equivalent to F that satisfies both conditions, and that this set contains no trivial FDs. Finally, we note that since **R** is independent under F, no FD of F can apply to two schemes in F. We will say that FD $X \rightarrow A$ in F is *from* R_i when R_i is the relation scheme in **R** such that $X A \subseteq R_i$.

Next, we introduce some variations on the rules for chasing with FDs. The normal F-rule for chasing an EI T under an FD $X \rightarrow A \in F$ takes two rows v and w in T with $v(X) = w(X)$ and tries to equate $v(A)$ with $w(A)$. In equating entries, we allow nulls to be replaced by values and lower-numbered nulls. If $v(A)$ and $w(A)$ are distinct values, and FD violation has occured, and we set T to ϕ. Also, in applying an F-rule, or its variants below, only nulls in v and w may be changed. We shall use two restricted forms of the F-rule in this proof. The NF-rule (null-preserving F-rule) will only equate $v(A)$ and $w(A)$ if one is a value and the other a null. It will not equate two nulls. The BF-rule (basic F-rule) is more

[2]E. Chan has idependently shown a similar result <Ch1, Ch2>.

restrictive than the NF-rule. The BF-rule is used when T is T_d, or derived from T_d by chasing. Let $X \rightarrow A$ be from R_i. The BF-rule requires that one of v and w come from $r_i(R_i)$ originally. Note that the row, say v, that came from r_i will have a value at $v(A)$, and that for some tuple t in r_i, $t(X\ A) = v(X\ A)$.

We now use the NF-rule and the BF-rule to define two restricted types of chase.

Definition: If T is an EI, then the *null-preserving chase* **of** T **under FDs** F**, denoted** $nchase_F(T)$**, is one in which only the NF-rule for FDs in** F **is used.**

Definition: If T is an EI that is derived from T_d, then the *basic chase* of T under FDs F, denoted $bchase_F(T)$, is one in which only the BF-rule for FDs in F is used.

We state without proof that $nchase_F(T)$ and $bchase_F(T)$ represent finite Church-Rosser processes, hence their results are unique. Using a set of FDs G equivalent to F could give different results, but we noted that F is uniquely determined by our assumptions. Also note that if either resticted chase uncovers an FD violation, so will $chase_F(T)$. Note that in both $nchase_F(T_d)$ and $bchase_F(T_d)$, when a rule for $X \rightarrow A$ is applied to rows v and w, both $v(X)$ and $w(X)$ will contain no nulls, since neither chase equates nulls.

Definition: The *restrained representative instance* for database d under F, denoted $RRI_F(d)$, is $bchase_F(T_d)$.

The following proposition shows that $nchase$ can be used in place of $bchase$ in computing $RRI_F(D)$.

Proposition 1: If d is a locally (hence globally) satisfying database state on **R**, then

Proof of Proposition 1: We can certainly compute $nchase_F(T_d)$ by first computing $T = bchase_F(T_d)$ and then computing $nchase_F(T)$. The proposition will be proved if we can show that $nchase_F(T) = T$. Suppose some NF-rule for $X \rightarrow A$ in

F applies to T, where $X \to A$ is from R_i. Say the rule applies to rows w_1 and w_2 to change $w_2(A)$ to $w_1(A)$ (hence $w_1(A)$ is a value and $w_2(A)$ a null). There cannot be a tuple u in $r_i(R_i)$ with $u(X) = w_1(X)$ (= $w_2(X)$), or else the BF-rule could have been used to fill in $w_2(A)$, as $w_1(A)$ would have to equal $u(A)$.

We construct a new database state d' from d by adding a tuple u' to r_i, where $u'(X) = w_1(X)$. Here is how we form u'. Let u be a tuple over R_i such that $u(X) = w_1(X)$ and u is distinct marked nulls on $R_i - X$. Chase $r_i(R_i) \cup \{u\}$ under the FDs in F that apply to R_i. (Any variety of chase will yield the same results here.) No FD violation arises, or else one would arise in $chase_F(T_d)$, showing that d is not globally satisfying. If this "mini-chase" does not fill in all the nulls in u, change the remaining nulls to new values that do not appear elsewhere in d. The resulting tuple is u'. Note that $u'(A)$ must be one of the new values. If $u(A)$ had been filled in with a value, it would have been filled in during $bchase_F(T_d)$.

We have been careful to construct u' so that d' is locally satisfying. All the relations except r_i are the same as in d, and r_i has only had u' added, which violates no FDs. Let w' be the row for u' in $T_{d'}$. Consider computing $bchase_F(T_{d'})$ by intially ignoring w'. We eventually obtain an EI $T' = T \cup \{w'\}$. Now consider: $w_1(X) = w'(X)$, but $w_1(A) \neq w'(A)$, since $w'(A) = u'(A)$ is a new value. Thus we have a violation of $X \to A$, and d' is not globally satisfying--a contradiction to the independence of $\mathbf{R}$ under F.

We conclude that no NF-rules can be applied to T, so $bchase_F(T)$ = $nchase_F(T)$.

Corollary: If in computing $RRI_F(d)$ for some d, we generate a row w where w is has no nulls on $X\,A$, and $X \to A$ is an FD in F from R_i, then there is a tuple $t \in r_i(R_i)$ with $t(X\,A) = w(X\,A)$.

Definition: For a row t in a RI or RRI, let $w down-arrow$ be the non-null portion of w.

Proposition 2: Let $d(\mathbf{R})$ be database satisfying F and let T be $RRI_F(T_d)$.

1. For any row w in T, $w down-arrow$ is in $E(d)$ for some FD-join expression E on $\mathbf{R}$.
2. For any tuple $t(S) \in E(d)$ on $\mathbf{R}$ there is a row w in T with $w(S) = t$.

Proof of Proposition 2: (Part 1.) The statement is true for T_d. We show it remains true after application of a BF-rule to change a null to a value. Assume $w down-arrow$ is in $E(d)$, for some FD-join expression E. Suppose $X \rightarrow A$ from R_i is used with row w' to change $w(A)$ to $w'(A)$. We noted in the last corollary that $r_i(R_i)$ must contain a tuple t with $t(X\,A) = w'(X\,A)$. Therefore, $E'(d)$ contains $w down-arrow$ after $w(A)$ is filled in, where $E' = E \bowtie \pi_{XA}(r_i)$.

(Part 2.) The statement is clearly true if E is just r_i. Suppose that $t \in E(d)$ where $E = E' \bowtie \pi_{XA}(r_i)$ and $X \rightarrow A \in F$. Thus, $t(S - A) \in E'(d)$. We inductively assume that there is a row w_1 in T with $w_1(S - A) = t(S - A)$. We also know that r_1 must have a tuple u with $u(X\,A) = t(X\,A)$. Let w_2 be the row in T_d coming from u. We can apply the BF-rule for $X \rightarrow A$ to w_1 and w_2 to set $W_1(A) = w_2(A)$. Thus, in T, $w_1(S)$ must equal $t(S)$, or else we can get and FD violation, and d is not globally satifying.

From Proposition 2 we can conclude that for any $X \subseteq U$, there is some expression E that is the union of projections of FD-joins on $\mathbf{R}$ such that $E(d) = \pi_x(RRI_F(d))$ for every satisfying state d. In particular, we can form E by taking every FD-join expression D (that doesn't repeat terms) where $schema(D) \supseteq X$, projecting each onto X, and taking the union.

The strategy for the rest of the proof is as follows. Given a database $d(\mathbf{R})$ satisfying F, we want to show that $RRI_F(d)$ contains all the combinations of values that $RI_F(d)$ does. ($RRI_F(d)$ and $RI_F(d)$ could differ in that $RI_F(d)$ could have equated nulls.) To do so, we exhibit a database d^* satisfying F that contains d (relation by relation) such that $RRI_F(d^*) = RI_F(d^*)$. Furthermore, there will be a mapping ψ from rows of $RRI_F(d^*)$ to rows of $RRI_F(d)$ such that w and $\psi(w)$ agree everywhere w has a value that appears in $RRI_F(d)$. We will construct d^* a step at a time, where a step adds one tuple to one relation in d. The addition will have the effect of "promoting" a null to a value in $RRI_F(d)$.

Let $d(R)$ satisfy F. Let $T = RRI_F(d)$. We know from Proposition 1 that no NF-rule can be applied to T. Suppose some F-rule for $X \rightarrow A$ can be applied to make changes in T. The F-rule must equate two nulls, since if it equates a null and a value, so could an NF-rule. Hence suppose the F-rule for $X \rightarrow A$ can be applied on rows v and w of T to equate nulls $v(A)$ and $w(A)$. We must have $v(X)$ and $w(X)$ free of nulls, since T has no repeated nulls. Let $X \rightarrow A$ be from R_i. There cannot be a tuple $t \in r_i$ with $t(X) = v(X)$, or else $v(A)$ and $w(A)$ would ahve been given the value $t(A)$ in computing T.

We shall use the same construction we used to form database d' in the proof of Proposition 1. We can add a tuple u' to r_i such that $u'(X) = v(X)$, $U'(A)$ is a new value found nowhere else in d, and $r_i \cup \{u'\}$ satisfies F. If X' is the set of attributes where u' has original (to d) values, then $v(X') = u'(X')$, as any values filled into u' during the "mini-chase" will also have been added to v during the computation of T.

Let us compare $T = RRI_F(d)$ to $T' = RRI_F(d')$. We can compute $RRI_F(d')$ by first computing $T_1 = RRI_F(d) \cup \{y\}$, where y is the padded version of u'. (That is, do nothing with the row for u' initially.) We know that $v(X) = w(X) = y(X)$, so we may continue by setting $v(A)$ and $w(A)$ to $y(A)$ $(= u'(A))$. We have, in effect, promoted the nulls in $v(A)$ and $w(A)$ in T to the value $y(A)$ in T'. We may also be able to use y to fill in new values for other nulls in T_1. Notice, however, that in computing T', we can ensure that v always *supersedes* y, in the sense that if

$y(B) = x(B)$ for some row $x \neq y$ in T_1, then $v(B) = y(B)$. Call the set of all such attributes $match(y)$. Initially, $match(y) = X'$, and $v(X') = y(X')$, so $v(match(y)) = y(match(y))$. If y is used to fill in a value for null $x(B)$ in any row x of T_1, $v(B)$ can be filled in with $y(B)$ first. If any row x is used to fill in a null $y(B)$, then either $v(B)$ can be filled in with $x(B)$ first, or $v(B)$ already equals $x(B)$. Thus, in T', $v(match(y)) = y(match(y))$.

What can happen to the rest of the rows in T in going from T_1 to T'? We will argue that no row other than y will have a null replaced by an original value, hence T and T' will have exactly the same combinations of original values in their rows. Consider continuing with $bchase_F$ from T_1. If a null $x(B)$ gets changed to a new value by a BF-rule for $Z \rightarrow B$, that value have come from $y(B)$, so $Z \rightarrow B$ is from R_i. This restriction follows from the corollary to Proposition 1 and the observation that u' is the only possible tuple in d' that contains new values.

Now consider some null in T_1 that gets filled in with an original value. We want to show that only y gets nulls replaced by original values. Suppose we use the BF-rule for $Z \rightarrow B$ on a row w_1 to fill in $w_2(B)$. If $W_1(Z\ B)$ is all original values, then $w_2(B)$ would have been filled in in T, unless w_1 or w_2 is y. If w_1 is y, then $Z\ B\ \subseteq R_i$, so v could have been used in its place with an NF-rule, as v has all the original values that y does. Since $bchase_F(T_d) = nchase_F(T_d)$ by Proposition 1, there is some way that $w_2(B)$ would have been filled in with a value in T. If w_2 is y, we don't care.

If, on the other hand, $w_1(Z\ B)$ contains some new values, then, following previous arguments, $Z \rightarrow B$ is from R_i and w_1 is y. If $y(B)$ is a new value, we don't care. If $y(B)$ is an original value, consider the following. Assume $w_2(B)$ is the first null to be filled in with an original value from y. Let Q be the maximal set of attributes such that $y(Q) = w_2(Q)$ in T_1. Observe that $y(Q)$ is all original values. Before applying the BF-rule for $Z \rightarrow B$ to w_2, if any of its nulls were filled in, they must have been replaced by new values from y. It follows that $Q \rightarrow Z$. Therefore $Q \rightarrow B$ must be in F. Now $y(Q\ B)$ is all original values and $Q\ B \subseteq R_i$, so v could be used to fill in $w_2(B)$ using the NF-rule for $Q \rightarrow B$. As argued before, $w_2(B)$ must have been filled in with a value in T.

So, the only nulls changed to original values in going from T to T' are in row y, and y is superseded by v. We can easily construct a mapping ψ from T' to T as described before: For a row $w \in T'$, everywhere w has an original symbol, $\psi(w)$ has the same symbol. Mapping ψ takes every row in T' other than y to the corresponding row in T, and takes y to v.

All this works has been but one step in converting d to d^* . Suppose in $RRI_F(d')$ an F-rule can be used to equate nulls. We can then form d'' by adding a row to some relation in d' to promote those nulls to values, with a mapping ψ' from $RRI_F(d'')$ to $RRI_F(d')$ that preserves combinations of original values. Note that $\psi' \circ \psi$ gives such a mapping directly from $RRI_F(d'')$ to $RRI_F(d)$. We can continue to add rows to relations in d to promote nulls that can be equated by F-rules. We obtain a sequence of database states d, d', d'', $d^{(3)}$, $d^{(4)}$,... . Do we ever reach a database state $d^{(i)}$ in this sequence where $RRI_F(d^{(i)}) = RI_F(d^{(i)})$? (State $d^{(i)}$ is the desired state d^* .) The answer is yes. Notice a newly added row in $RRI_F(d^{(j)})$ is superseded by some row in $RRI_F(d^{(j-1)})$. By induction, the new row is superseded by some row in $RRI_F(d)$. Therefore, the number of new rows added to d is bounded by the number of nulls in $RRI_F(d)$, and d^* will be reached eventually.

End of Proof of Theorem 1

Maier, Ullman and Vardi <MUV> suggest a departure from the two-step paradigm for universal scheme query processing that can be used with representative instances. Rather than apply a query to the intersection of projections of weak instances, apply it to the projections individually, and then intersect the results. For the alternative model to be attractive, there must be an effective method for computing the intersection of query results, of course.

3. Properties and Theory of Window Functions

Here we look at several properties of window functions, and see which window functions defined so far have those properties. We feel that the first two properties given, the containment condition and faithfulness, are minimum conditions for a reasonable window function.

3.1. The Containment Condition

By URA, a set of attributes uniquely determines a connection among the attributes themselves, which connection a window is supposed to transmit. If a set of attributes X is a subset of Y, whatever the connection among the attributes of X, it must be an aspect of the connection among the attributes of Y.

Example 9: Under URA, it is permissible for [S C I] to mean a student takes a course from an instructor and for [S C] to mean a student takes a course. URA would not be satisfied if the meaning of [S C I] were changed to a student is a TA for a course under an instructor.

For a window function to be consistent with URA, whenever $X \subseteq Y$ and t is a tuple in $[Y]$, $t(X)$ should be in $[X]$. Stated another way, $[X] \supseteq \pi_X([Y])$. This inequality is the *containment condition*. It is similar to Sciore's notion of *downward closure* <Sc1>.

In the two systems that do not require a URA database, APPLE and q, the window functions cannot be shown to necessarily satisfy the containment condition for all database states. Even if virtual relations in q are defined solely by joins, it is not sufficient to guarantee the containment condition is satisfied.

Lemma 1: $[\cdot]_I$ satisfies the containment condition.

Proof: For the universal instance window function, we have a stronger condition, namely $[X]_I = \pi_X([Y]_I)$ for $X \subseteq Y$.

Lemma 2: Let $\{s_1(S_1), s_2(S_2), \ldots, s_m(S_m)\}$ be a set of relations where $S_1 S_2 \cdots S_m \supseteq X$. Let $\{q_1, q_2, \ldots, q_n\}$ be a set of relations that includes $\{s_1, s_2, \ldots, s_m\}$. Then

$$\pi_X(s_1 \bowtie s_2 \bowtie \cdots \bowtie s_m) \supseteq \pi_X(q_1 \bowtie q_2 \bowtie \cdots \bowtie q_n).$$

Corollary: $[\cdot]_{LJ}$, $[\cdot]_{SJ}$, $[\cdot]_{\mathbf{M}}$, and $[\cdot]_{\mathbf{AO}}$ satisfy the containment condition.

Corollary: $[W]_{\mathbf{AO}} = r'(W)$, for $W \in \mathbf{O}$.

Lemma 3: Any weak instance window function satisfies the containment condition.

Proof: Let $X \subseteq Y$ and let $\mathbf{C}$ be the set of dependencies for the window function.

$$[X]_{\mathbf{C}} = \bigcap_{I \text{ a } \mathbf{C}\text{-WI}} \pi_X(I) = \bigcap_{I \text{ a } \mathbf{C}\text{-WI}} \pi_X(\pi_Y(I)) \supseteq$$
$$\pi_X\Big(\bigcap_{I \text{ a } \mathbf{C}\text{-WI}} \pi_Y(I)\Big) = \pi_X([Y]_{\mathbf{C}}).$$

3.2. Faithfulness

The principle for the next condition is "What you see is what you've got." The containment condition requires that the set of views given by a windoe function be consistent with each other. The views given by a window function should also be consistent with the contents of the database. A window function is *faithful* if for any relation scheme $R \in \mathbf{R}$, the relation on R in the database agrees with the window on R, for all states of the database: $r(R) = [R]$.

This definition assumes that the database has no two relations with the same scheme, which will be true if the database satisfies URA. Since a database in APPLE could have two relations on the same scheme, the definition does not apply there. A window function for q will be faithful if the convention is followed that there is only one relation per scheme, and stored relations come before virtual relations in the relfile.

The universal instance window function is clearly faithful. The window function $[\ ]_{SU}$ is not necessarily faithful. Consider the database scheme $\mathbf{R} = \{A\ B,\ B\ C,\ A\ C\}$, with B a key of $B\ C$. The window $[A\ C]$ will not necessarily agree with $r(A\ C)$, since $[A\ C]$ contains tuples from $\pi_{AC}(A\ B \bowtie B\ C)$.

The following lemma assumes that there are not two distinct relations schemes R and S in $\mathbf{R}$ such that $R \subseteq S$. If $\mathbf{R}$ contains such schemes, the lemma holds if the database relations satisfy the containment condition.

Lemma 4: $[\cdot]_{SU}$ and $[\cdot]_{\mathbf{H}}$ are faithful.

Proof: Under tableau minimization, $\pi_R(r_1 \bowtie r_2 \bowtie \ldots \bowtie r_p)$ will always be reduced to r_i, where r_i is the relation with scheme R.

Theorem 2: $[\cdot]_{\mathbf{AO}}$ is faithful if and only if $\mathbf{A} \subseteq \mathbf{O}$ and the relations on associations in $\mathbf{A}$ satisfy the containment condition.

Proof: In the proof, $[\cdot]$ will mean $[\cdot]_{\mathbf{AO}}$.

(only if) We show the contrapositive. Let R be an association of $\mathbf{A}$ that is not in $\mathbf{O}$. Consider a state of the database where $r(R) \neq \phi$ and $r(S) = \phi$, $S \neq R$. If $\mathbf{O}$ has no object containing R, then $[R] = \phi$, and the window fucntion is not faithful to $r(R)$. If $W \in \mathbf{O}$ and $W \supseteq R$, the join used to form $r'(W)$ must include at least one relation apart from $r(R)$. Hence $r'(W) = \phi$. This equality holds for any object containing R, so $[R] = \phi \neq r(R)$.

Now suppose $\mathbf{A} \subseteq \mathbf{O}$, but the database relations do not satisfy the containment condition. Let R and S be associations in $\mathbf{A}$ such that $R \subseteq S$, but $r(R) \not\supseteq \pi_R(r(S))$. It follows that $r(R) \bowtie r(S)$ is not a proper subset of $r(S)$. For any object W, $W \supseteq S$, $r(R)$ and $r(S)$ will enter the join for $r'(W)$, so $\pi_S(r'(W)) \subseteq r(R) \bowtie r(S)$. Hence, $[S]$ is properly contained in $r(R) \bowtie r(S)$, and is not faithful to $r(S)$.

(if) If $\mathbf{A} \subseteq \mathbf{O}$, and the containment condition holds for the database relations, it is not hard to show that $r(R) = r'(R)$ for any $R \in \mathbf{A}$. It follows that $[R] \supseteq r(R)$. By Lemma 2, for any object W containing R, $\pi_R(r'(W)) \subseteq r'(R)$, so $[R] \subseteq r(R)$, and we have the desired equality.

Henceforth, when dealing with association-object window functions, we shall assume $\mathbf{A} \subseteq \mathbf{O}$ and that the relations on associations in $\mathbf{A}$ satisfy the containment condition.

Weak instance window functions are not necessarily faithful. For FDs, Mendelzon <Me> shows that for a database state d, there is a complete state d' with the same set of weak instances. A *complete* state essentially is one that is the projection of its representative instance. Weak instance window functions are faithful on complete database states. We also have the following two theorems about particular weak instance window functions.

Theorem 3: $[\cdot]_{\mathbf{K}}$ is faithful if every relation scheme in **R** has a nontrivial key.

Sketch of Proof: Consider a locally satisfying (hence globally satisfying) database state d. Let $R \in \mathbf{R}$ be a relation scheme such that π@*down-arrow*@$_X(RI_{\mathbf{K}}(d))$ contains a tuple t not in $r(R)$. Let K be a nontrivial key for R. Modify d to d' by adding t' to $r(R)$, where $t'(K) = t(K)$, but $t'(R-K) \neq t(R-K)$. Since r(R) does not already have a tuple that agrees with t on K, d' is locally satisfying. However, d' is not globally satisfying, since t will still show up as part of a row of $RI_{\mathbf{K}}(d')$ and contradict t'.

The windows in $[\cdot]_{\mathbf{K}}$ can be unfaithful if **R** has relations with only trivial keys. Let $R = \{A\ B, A\ C, B\ D, C\ D\}$, with key FDs $A \rightarrow C$ and $B \rightarrow D$. Then the expression $\pi_{CD}(r(A\ B) \bowtie r(A\ C) \bowtie r(B\ D))$ can add tuples to $[C\ D]_{\mathbf{K}}$ that are not in $r(C\ D)$. Note that a relation scheme formed by synthesis <Be> will have no trivial keys. Theorem 3 also holds if the only scheme with a trivial key is a universal key <BDB>.

Theorem 4: $[\cdot]_{*\mathbf{R}}$ is faithful.

Sketch of Proof: Look at $R \in \mathbf{R}$ and consider the computation of $R/_{*\mathbf{R}}(d)$. The JD $*R$ is the only dependency used in chasing T_d when forming the representative instance for d. We can show by induction that at each stage of the computation of the representative instance, if any row w is non-null on R, then $w(R) \in r(R)$. Also, for any tuple $t \in r(R)$, there will always be a row w with $w(R) = t$. We conclude that $[R]_{*\mathbf{R}} = r(R)$.

3.3. Integrity of Objects

The purpose of the next condition is to prevent a little knowledge from being a dangerous thing. The condition is stated in terms of objects, so it applies to only association-object window functions. In this subsection, $[\cdot]$ will mean $[\cdot]_{\mathbf{AO}}$. In Section 3.4, we show how objects can be defined on any window function, so we shall be able to apply the condition more generally.

The idea behind integrity of objects is that if someone knows the semantics of all the associations within an object W, then he should be able to deduce the meaning of the connection on any subset of W. Formally, for $W \in \mathbf{O}$, let

$$a(W) = \{R \in \mathbf{A} | R \subseteq W\}.$$

Object W is *integral* relative to $[\cdot]$, if for any subset X of W, $[X]$ can be computed from $\{r(R) | R \in a(W)\}$.

Example 10: To se how integrity of objects can fail, consider U = {P (painting), O (owner), R (artist), D (address)}, $\mathbf{A}$ = {P O, P R, O D, R D}, and $\mathbf{O} = \mathbf{A} \cup$ {P O D, P R D}. We are storing information on owners and artists of paintings, and addresses of owners and artists, and making connections on owners and artists. In the object P O D, the connection from painting to address is via owner. However, the object P R D can also add tuples to [P D], so [P D] cannot be computed from relations in α(P O D) alone. The danger here is that if a user knows that the database has information about paintings, owners and addresses, but does not know about artists, his assumption as to the meaning of [P D] will be incorrect. The window [P D] is really the combination of two different connections.

The next theorem shows that integrity of objects is equivalent to the objects being closed under nonempty intersection. This closure property has a computational advantage. It implies that for any X, there is a unique minimal object W containing X. That is, for any other object V that contains X, $V \supseteq W$. Thus, $[X]$ can be computed as $\pi_X(r'(W))$, since for any object $V \supseteq X$,

$$\pi_X(r'(V)) \subseteq \pi_X(r'(W)).$$

No unions need be taken to compute $[X]$.

Theorem 4: All objects in **O** are integral if and only if **O** is closed under nonempty intersection.

Proof: (if) By the remarks above, if W is an object and $W \supseteq X$, there is a minimal object W', $W \supseteq W' \supseteq X$, such that $[X] = \pi_X(r'(W'))$. The object relation $r'(W')$ depends on only relations for $\alpha(W')$, which is a subset of $\alpha(W)$. Hence, W is integral.

(only if) Let X be the intersection of objects V and W, where X is not itself an object. Assume no objects smaller than V and W have intersection X. There must be some association R in $\alpha(W)$ such that R is not a subset of V, so R is not in $\alpha(V)$. By considering states of the database that differ by $r(R)$ being empty or nonempty, it is possible to induce changes in $[X]$ that do not depend on relations for $\alpha(V)$. Therefore, V is not integral.

There are direct arguments that the closure of objects under intersection is desirable. With closure under intersection, any window takes its value from a single object. There are no unions needed to compute windows. When unions are used, there is always the danger that the user is aware of only one or some of the connections used to compute a window.

Multiple connections in a window may not be a big problem if the various connections are of the same "flavor," as in the [B C] window in the banks example. There is a common generalization of the two connections involved, namely "customer does business with the bank." In the paintings example, there is no natural generalization of the two connections between paintings and addresses via owner and artist, since the association between paintings and owners has quite a different flavor from the association between artists and paintings.

Even if objects are not closed under intersection, it seems that associations should be. Consider associations R_1 and R_2 whose intersection is S. It makes sense to have an S-value without any values from $R_1 - S$ (in an R_2-tuple). Likewise, we can have an S-value without values from $R_2 - S$. It seems that we should be able to store S-values with neither values from $R_1 - S$ nor $R_2 - S$, so S should be an association.

There are at least two ways to modify a set of objects to get closure under intersection. One method is adding more objects and the other method is renaming attributes. The first method is probably better for the banks example: add an association B C, meaning the customer deals with the bank. In the paintings example, the second method is preferable: rename D (address) to OWNER_D and ARTIST_D.

3.4. Implicit Objects

While objects were used in the definition of only one of the window functions, we can pick out sets of attributes that behave as objects relative to other window functions. V is an *implicit object* for a window function [ ] if there is some state of the database where the inclusion

$$[V] \supseteq \pi_V(\bigcup_{W \supset V} [W])$$

is strict. (The inclusion always holds if the window function satisfies the containment condition.) That is, $[V]$ can contain a tuple that is not in the projection of any window on a scheme larger than V.

It is not hard to show for $[\cdot]_{\mathbf{AO}}$ that the implicit objects are precisely **O**. For other window functions, especially weak instance window functions, it is useful to discriminate objects from non-objects. We need only store expressions for the windows on implicit objects in order to have a simple means to compute all the windows. For the two specific weak instance window functions we covered, we can characterize the implicit objects.

Theorem 5: V is an implicit object for $[\cdot]_{\mathbf{K}}$ if V is the union of relation schemes that have a lossless extension join under **K**.

Theorem 6: V is an implicit object for $[\cdot]_{*\mathbf{R}}$ if V is the scheme of an embedded join dependency $*\mathbf{S}$ implied by $*\mathbf{R}$ where $\mathbf{S} \subseteq \mathbf{R}$.

Proof: The result follows from two facts.

1. If $RI_{*\mathbf{R}}(d)$ contains a row that is non-null exactly on V, then there is an embedded JD implied by $*\mathbf{R}$ with scheme Y. (Lemma 5.1 of Yannakakis <Ya>.)

2. If Y is the scheme of and embedded JD implied by $*\mathbf{R}$, then it is possible to find a database state d such that $RI_{*\mathbf{R}}(d)$ contains a row defined exactly on V, and no rows that are non-null on more than V.

Both theorems imply that all relation schemes are implicit objects. Using hypergraph notation <BFMY>, we can describe the implicit objects for $[\cdot]_{*\mathbf{R}}$. The JD $*\mathbf{R}$ implies the embedded JD $*\mathbf{S}$, $\mathbf{S} \subseteq \mathbf{R}$, if and only if **S** is closed, connected, and whenever it contains two edges of a block of **R**, it contains all the edges in the block.

The definition of an object being integral can be extended to any window functino by phrasing it in terms of implicit objects and defining $\alpha(W)$ in terms of

R

Lemma 5: $[\cdot]_{\mathbf{K}}$ does not guarantee integrity of objects.

Proof: Let $\mathbf{R} = \{A\ B\ C,\ B\ C\ D,\ A\ D\ E\}$ and keys $A\ B$, $B\ C$ and $A\ D$. (This example is due to Sagiv <Sa1>.) The expression for $[A\ D]_{\mathbf{K}}$ is

$$\pi_{AD}(r(A\ D\ E)) \cup \pi_{AD}(r(A\ B\ C) \bowtie r(B\ C\ D)).$$

$A\ D\ E$ is an implicit object containing $A\ D$, but $[A\ D]_{\mathbf{K}}$ depends on more than $r(A\ D\ E)$.

Lemma 6: $[\cdot]_{*\mathbf{R}}$ guarantees integrity of objects.

Proof: If V and W are schemes of embedded JDs implied by $*\mathbf{R}$, then there is an embedded JD on scheme $V \cap W$. Thus, implicit objects for $[\cdot]_{*\mathbf{R}}$ are closed under intersection, and Theorem 4 applies.

4. Further Work

One objection to URA is that some connections may be lost if attributes are renamed in order to satisfy it. Suppose we have two FACULTY-STUDENT relationships: FACULTY has a STUDENT in a COURSE, and a FACULTY advises a STUDENT. We can rename FACULTY to INSTRUCTOR and ADVISOR to distinguish the two roles. However, we lose the connection between INSTRUCTOR and ADVISOR, and with properties of FACULTY, such as OFFICE. To address this loss, an explicit hierarchy of roles can be introduced <Sc2, SS>. We have been looking at extensions to the association-object window function that allow equijoins on attributes related by the role hierarchy <MRS>. For example, we can use the equijoin on ADVISOR = FACULTY of the relations r(STUDENT ADVISOR) and r(FACULTY OFFICE) to connect a STUDENT to his or her ADVISOR's OFFICE. Beeri and Korth <BK> describe a similar approach that involves FD information as well. Sciore and Warren <Sc3> have been experimenting with "file grammars," which allow windows to include multiple instances of the same attribute, as might arise in computing an EMPLOYEE's MANAGER's MANAGER from the relation r(EMPLOYEE MANAGER).

We consider the question of when a weak instance window function had an equivalent computational definition. The dual question is whether every computational window function has an equivalent weak instance definition. An association-object window function need not have a weak instance definition. The problem comes in that weak instance definitions assume every tuple in the database is part of some universal tuple over U, where in the association-object model, U might not be an object. That is, some attributes can be too semantically distant to be connected automatically. Recent work on extending weak instances to have "placeholder" nulls <La, St, U12> should allow a weak instance definition for association-object window functions. The affect on a representative instance is to have "non-chaseable" nulls initially and use *existence constraints* <Ma2> to indicate where "chaseable" nulls may be inserted.

Finally, we note that we been considering windows as purely a mechanism for database query. What about update? The does not seem to be the flexibility to update over arbitrary schemes that there is to query over arbitrary schemes. The work that has been done on universal scheme update <St> indicates that either a user must be prompted to supply values on additional attributes, or that the database must store "missing value" nulls.

5. Bibliography

This paper is a revised and expanded version of an earlier conference abstract <MRW>.

<AK> A.V. Aho, B.W. Kernighan. research!user/ava/q/README, 1980.

<ABU> A.V. Aho, C. Beeri, J.D. Ullman. The theory of joins in relational databases, *ACM TODS* 4(3), September 1979, 297-314.

<ASU1> A.V. Aho, Y. Sagiv, J.D. Ullman. Equivalence of relational expressions, *SIAM J. on Computing* 8(2), May 1979, 218-246.

<ASU2> A.V. Aho, Y. Sagiv, J.D. Ullman. Efficient optimization of a class of relational expressions, *ACM TODS* 4(4), December 1979, 435-454.

<A-G> Z. Arazi-Gonczarowski. A high-level interface for users in a relational database, manuscript, Dept. of Computer Science, Hebrew University, 1983.

<AC> A.K. Arora, C.R. Carlson. The information preserving properties of relational database transformations, VLDB IV, October 1978, 352-359.

<Ba> E. Babb. Joined normal form: A storage encoding for relational databases, *ACM TODS* 7(4), December 1982, 588-614.

<BFMMUY> C. Beeri, R. Fagin, A.O. Mendelzon, D. Maier, J.D. Ullman, M. Yannakakis. Properties of acyclic database schemes, Thirteenth ACM Symp. on Theory of Computing, May 1981, 355-362.

<BFMY> C. Beeri, R. Fagin, D. Maier, M. Yannakakis. On the desirable properties of acyclic database schemes, *JACM.* 30(3), July 1983, 479-513.

<BK> C. Beeri, H.F. Korth. Compatible attributes in a universal relation, ACM Symp. on Principles of Database Systems, March 1982, 55-62.

<BMSU> C. Beeri, A.O. Mendelzon, Y. Sagiv, J.D. Ullman. Equivalence of relational database schemes, *SIAM J. on Computing* 10(2), May 1981, 352-370.

<Be> P.A. Bernstein. Synthesizing third normal form relations from functional dependencies, *ACM TODS* 1(4), December 1976, 277-298.

<BB> J. Biskup, H.H. Bruggeman. Universal relation views: a pragmatic approach, VLDB IX, October-November 1983, 172-185.

<BDB> J. Biskup, U. Dayal, P.A. Bernstein. Synthesizing independent database schemas, 1979 ACM SIGMOD Conf., May-June 1979, 143-152.

<CK> C.R. Carlson, R.S. Kaplan. A generalized access path model and its application to a relational database system, 1976 ACM SIGMOD Conf., June 1976, 143-154.

<Ch1> E.P.F. Chan. Optimal computation of X-total projections for independent schemes, manuscript, CSRG, Univ. of Toronto, March 1983.

<Ch2> E.P.F. Chan. Efficient optimization of unions of simple chase join expressions, to be presented at 1984 ACM SIGMOD Conf., June 1984.

<FMU> R. Fagin, A.O. Mendelzon, J.D. Ullman. A simplified universal relation assumption and its properties, *ACM TODS* 7(3), September 1982, 343-360.

<Gr> M.H. Graham. On the universal relation, CSRG Report, Univ. of Toronto, December 1979.

<Ho1> P. Honeyman. Extension Joins, VLDB VI, October 1980, 239-244.

<Ho2> P. Honeyman. Testing satisfaction of functional dependencies, *JACM* 29(3), July 1982, 668-677.

<Ho3> P. Honeyman. Finding lossless joins, manuscript, December 1982.

<Ko> System/U: a progress report, XP2 Workshop on Relational Database Theory, June 1981.

<KKU> H.F. Korth, G.M. Kuper, J.D. Ullman. System/U: A database system based on the universal relation assumption, Stanford Computer Science technical report 82-944, January 1983.

<KU> H.F. Korth, J.D. Ullman. System/U: A database system based on the universal relation assumption, XP1 Workshop on Relational Database Theory, June-July 1980.

<KMRS> S.M. Kuck, D.A. McNabb, S.V. Rice, Y. Sagiv. The Parafrase database user's manual, Computer Science Technical Report 80-1046, Univ. of Illinois, December 1980.

<KS> S.M. Kuck, Y. Sagiv. A universal relation database system implemented via the network model, ACM Symp. on Principles of Database Systems, March 1982, 147-157.

<La> K. Laver. No-information nulls and regions of the universe, manuscript, January 1984.

<Li> Y.E. Lien. On the equivalence of database models, *JACM* 29(2), April 1982, 333-362.

<Lo> E.L. Lozinskii. Construction of relations in relational databases, *ACM TODS* 5(2), June 1980, 208-224.

<Ma1> D. Maier. Discarding the universal instance assumption: preliminary results, XP1 Workshop on Relational Database Theory, June-July 1980.

<Ma2> D. Maier. *The Theory of Relational Databases*, Computer Science Press, 1983.

<MMSU> D. Maier, A.O. Mendelzon, F. Sadri, J.D. Ullman. Adequacy of decompositions of relational databases, *J. of Computer and System Sciences* 17(2), December 1980, 368-379.

<MMS> D. Maier, A.O. Mendelzon, Y. Sagiv. Testing implications of data dependencies, *ACM TODS* 4(4), December 1979, 455-469.

<MRSSW> D. Maier, D. Rozenshtein, S.C. Salveter, J.Stein, D.S. Warren. Toward logical data independence: A relational query language without relations, 1982 ACM SIGMOD Conf., June 1982, 51-60.

<MRS> D. Maier, D. Rozenshtein, J. Stein. Representing roles in universal scheme interfaces. Proceedings of the 1984 IEEE International Conference on Data Engineering, April 1984, 133-142.

<MRW> D. Maier, D. Rozenshtein, D.S. Warren. Windows on the World, 1983 ACM SIGMOD Conf., May 1983, 68-78.

<MU> D. Maier, J.D. Ullman. Maximal objects and the semantics of universal relation databases, *ACM TODS* 8(1), March 1983, 1-14.

<MUV> D. Maier, J.D. Ullman, M.Y. Vardi. The equivalence of universal relation definitions, to appear *ACM TODS*.

<MW> D. Maier, D.S. Warren. Specifying connections for a universal relation scheme database, 1982 ACM SIGMOD Conf., June 1982, 1-7.

<Me> A.O. Mendelzon. Database states and their tableaux, XP2 Workshop on Relational Database Theory, June 1981. To appear *ACM TODS*.

<Os> S.L. Osborn. Towards a universal relation interface, VLDB V, October 1979, 52-60.

<Ri> J. Rissanen. Independent components of relations, *ACM TODS* 2(4), December 1977, 317-325.

<Ro> D. Rozenshtein. Query and Role-Playing in the Association-Object Database Model, Ph.D. Thesis, SUNY at Stony Brook, June 1983.

<Sa1>	Y. Sagiv. Can we use the universal instance assumption without using nulls?, 1981 ACM SIGMOD Conf., April-May 1981, 108-120.
<Sa2>	Y. Sagiv. A characterization of globally consistent databases and their correct access paths, *ACM TODS* 8(2), June 1983, 266-286.
<Sc1>	E. Sciore. The universal instance and database design, Doctoral dissertation, Princeton Univ., October 1980.
<Sc2>	Improving semantic specification in a relational database, 1979 ACM SIGMOD Conf., May-June 1979, 170-178.
<Sc2>	E. Sciore. File grammars, XP4.5 Workshop, Stanford, CA, August 1983.
<SP>	K.L. Schenk, J.R. Pinkert. An algorithm for servicing multi-relational queries, 1979 ACM SIGMOD Conf., August 1977, 10-20.
<SS>	J.M. Smith, D.C.P. Smith. Database abstractions: Aggregation and generalization, *ACM TODS* 2(2), June 1977, 105-133.
<So>	J.F. Sowa. Conceptual graphs and data base interface, *IBM Journal of Research and Development* 20(4), July 1976, 336-357.
<St>	J. Stein. Data Definition and Update in the Association-Object Data Model. Ph.D. Thesis in preparation.
<Su>	K. Subieta. Navigational facilities for relational database, *Information Systems* 8(1), 1983, 29-36.
<Ul1>	J.D. Ullman. The U.R. strikes back. ACM Symp. on Principles of Database Systems, March 1982, 10-22.
<Ul2>	J.D. Ullman. Universal relation interfaces for database systems. IFIP 83, September 1983.
<Va>	Y. Vassiliou. Functional dependencies and incomplete information, VLDB VI, October 1980, 260-269.
<Wa>	A. Walker. A universal table relational data base model with blank entries, unpublished manuscript, 1979.
<Ya>	M. Yannakakis. Algorithms for acyclic database schemes, VLDB VII, September 1981, 82-94.
<Za>	C. Zaniolo. The database language GEM, 1983 ACM SIGMOD Conf., May 1983, 207-218.