Tree-structured object creation in database transformations

Jan Van den Bussche

August 27, 2007

Abstract

Within the class of "determinate" object-creating database transformations, identified by Abiteboul and Kanellakis, a natural proper subclass consists of the so-called "constructive" transformations. A determinate transformation is constructive if its input-output pairs satisfy a condition discovered by Jan Paredaens. In this note we point out that when object creation is "tree-structured," as is the case in tree-structured data models such as XML, determinate transformations are *always* constructive.

Dedicated to Jan Paredaens
for his 60th birthday

1 Introduction

In tree-structured data models, such as in the XML data model [16, 1], it is desirable that transformations can be expressed that are *object-creating*, meaning that the result of a transformation can contain objects (in this case, tree nodes) that do not appear in the input database. Indeed, the standard XML query language XQuery [17] allows the expression of object-creating queries by means of the element construction operation.

Well before the rise of the XML data model, however, general database transformations (possibly object-creating, and possibly non-deterministic) were already studied by Abiteboul and Vianu [4, 5], and Abiteboul and Kanellakis [3] introduced the class of "determinate" transformations as those that are non-deterministic only in the choice of the id's of the new objects. Abiteboul and Kanellakis also introduced a very natural query language, called IQL, for expressing general determinate transformations. IQL is equivalent to the relational algebra extended with three programming constructs: object creation; assignment to relation variables; and while-loops. At around the same time, another equivalent language, called GOOD, was introduced by Jan Paredaens and his collaborators [11].

Not all determinate transformations are expressible by an IQL program, however. There even exist single input-output pairs of database instances that

cannot be realized by any IQL program. This situation motivated Jan Paredaens to formulate a condition on pairs (I, J) of database instances that is necessary and sufficient for J to be an output of some GOOD (or IQL) program applied to I [7]. This condition states the existence of an extension homomorphism from the group of automorphisms of I to the group of automorphisms of J, and generalizes to object creation Jan Paredaens's earlier result on the BP-completeness (Bancilhon-Paredaens) of the relational algebra [14, 8, 10]. Later, the naturalness of the extension homomorphism condition was confirmed when it was shown that the IQL-expressible transformations are precisely those determinate transformations all of whose input-output pairs admit an extension homomorphism [15].

The developments just described happened largely before the rise of the XML data model. In this note, we ask ourselves what happens when object creation is tree structured, i.e., the newly created objects in the output form a tree, the leafs of which are labeled by objects from the input (a precise definition will be given later). XML is clearly tree-structured. We will show that in that case, the gap between determinate and IQL-expressible vanishes, i.e., the extension homomorphism condition is always satisfied for tree-structured object creation.

2 Database transformations

We recall some essential definitions in this section, largely taken from our earlier paper [15]. For background and motivation for these definitions, see Abiteboul, Hull and Vianu [2].

It is assumed that an infinite collection of *relation names* is given. To each relation name R a natural number $\alpha(R)$ is associated, called the *arity* of R, such that each number is the arity of infinitely many relation names. A *database schema* is a finite set of relation names.

It is furthermore assumed that a countably infinite universe $\mathbf{U}$ of abstract data elements, called *objects*, is given.

An *instance* I of a database schema $\mathcal{S}$ is a finite relational structure of type $\mathcal{S}$, consisting of a finite subset $|I|$ of $\mathbf{U}$, called the *domain*, and a mapping on $\mathcal{S}$, assigning to each relation name R of $\mathcal{S}$ a relation R^I on $|I|$ of rank $\alpha(R)$ (i.e., a subset of $|I|^{\alpha(R)}$), called the *content* of R. The set of all database instances of the schema $\mathcal{S}$ is denoted by $\mathrm{inst}(\mathcal{S})$.

Let $\mathcal{S}_{\mathrm{in}}$ and $\mathcal{S}_{\mathrm{out}}$ be two database schemas. A *determinate transformation from* $\mathcal{S}_{\mathrm{in}}$ *to* $\mathcal{S}_{\mathrm{out}}$ is an input-output relationship $Q \subseteq \mathrm{inst}(\mathcal{S}_{\mathrm{in}}) \times \mathrm{inst}(\mathcal{S}_{\mathrm{out}})$ satisfying the following three properties:

1. If $Q(I, J)$ then $|I| \subseteq |J|$;
2. If $Q(I, J)$ and f is a permutation of $\mathbf{U}$, then also $Q(f(I), f(J))$, where by $f(I)$ we mean the database instance obtained from I by applying f pointwise to all objects occurring in I;
3. If $Q(I, J_1)$ and $Q(I, J_2)$, then $J_2 = f(J_1)$ for some permutation f of $\mathbf{U}$ that is the identity on $|I|$.

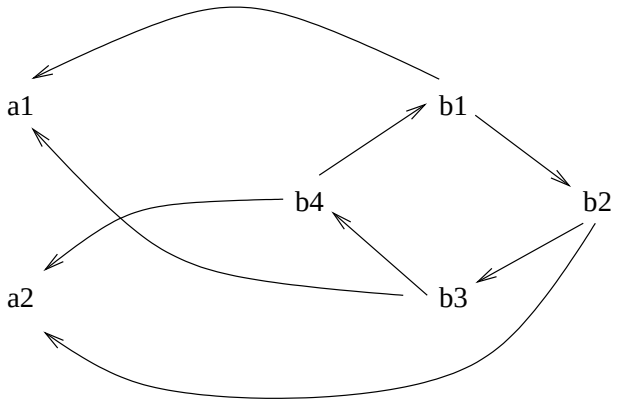

Figure 1: Instance I consists just of the two a's. Instance J adds the four b's together with the arrows (stored in a binary relation). There does not exist an extension homomorphism from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$.

The first requirement above is technical but harmless. The second one is a classical consistency criterion [10, 6] known as *genericity* [13]. The third, finally, expresses the determinacy property [3].

For a database instance I, we denote by $\mathrm{Aut}(I)$ the set of all permutations f of $|I|$ for which $f(I) = I$; such permutations are called automorphisms of I. The set $\mathrm{Aut}(I)$, with the operation of composition, forms a group structure.

A crucial concept, introduced by Jan Paredaens [7], now is the following. Let I and J be database instances such that $|I| \subseteq |J|$. An *extension homomorphism* from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$ is a group homomorphism $h : \mathrm{Aut}(I) \to \mathrm{Aut}(J)$ such that for each $f \in \mathrm{Aut}(I)$, the permutation $h(f)$ is an extension of f, i.e., $h(f)$ agrees with f on $|I|$.

We now call a determinate transformation Q *constructive* [15] if for every input-output pair (I, J) of Q, there exists an extension homomorphism from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$. An example (due to Serge Abiteboul [3]) of a pair of instances (I, J) that does *not* admit an extension homomorphism is shown in Figure 1. As a consequence, no transformation that contains (I, J) as an input-output pair can be constructive.

3 Tree-structured transformations

Note that the output of the non-constructive example from Figure 1 has an intrinsic cyclicity to it. That observation, and the recent interest in tree-structured data models such as XML, motivates us to study tree-structured transformations as a special class of determinate transformations. We first define this class formally and then prove that tree-structured determinate transformations are always constructive.

Definition 1 *Let J be an instance of some database schema $\mathcal{S}$, and let $T \in \mathcal{S}$ be a binary relation name. We call J* tree-structured by T *if J has the following two properties:*

1. *Let V equal the set of objects occurring in T^J, and consider this binary relation T^J as a directed graph on vertex set V. Then T^J must look like a set of rooted trees; more specifically, every vertex must have at most one incoming edge, and there must be no cycles.*

2. *For every relation name $R \in \mathcal{S}$ different from T, every tuple in the relation R^J must contain at most one occurrence of a vertex, i.e., an object from V.*

The first property in the above definition is, we hope, intuitive. The intuition behind the second property is that the tuples in the relations other than T serve as "annotations" or "labels" for the various tree vertices. We can formalize labels as follows:

Definition 2 *Let J be a database instance, tree-structured by T. Let x be a vertex of J. A* label *of x is any triple of the form $(R, i, \hat{t})$, where*

- *R is a relation name of J's database schema, with $R \neq T$;*
- *t is a tuple in R^J in which x appears;*
- *i is the position in T where x appears; and*
- *$\hat{t}$ is the subtuple of t obtained by omitting x.*

When two T-vertices have precisely the same set of labels, we call them duplicates. *We call J* duplicate-free *if there are no duplicate leafs in J, where a leaf is a vertex without outgoing edges in T^J.*

Our central notion is now the following:

Definition 3 *Let Q be a determinate transformation from $\mathcal{S}_{\text{in}}$ to S_{out}, and let T be a binary relation name in S_{out}. We call Q* tree-structured by T *if for every input-output pair (I, J) of Q, the output J is tree-structured by T, with vertex set equal to $|J| - |I|$ (i.e., the set of newly created objects), and J is also duplicate-free.*

The requirement that J be duplicate-free is mainly for technical reasons. A transformation that is not duplicate-free can be easily made so by adding additional auxiliary nodes and labels.

The purpose of this note is to point out the following:

Theorem 1 *Every tree-structured determinate transformation is constructive.*

4 Proof

To prove the theorem, we recall some further definitions [15].

Let D be a subset of $\mathbf{U}$. The set $\mathrm{HF}(D)$ of *hereditarily finite sets (HF-sets) with ur-elements in D* [9] is the smallest set with the property that each finite subset of $D \cup \mathrm{HF}(D)$ is itself an element of $\mathrm{HF}(D)$.

An *HF-instance* I is defined as an ordinary instance, the only difference being that the domain $|I|$ is a subset of $\mathbf{U} \cup \mathrm{HF}(\mathbf{U})$ instead of $\mathbf{U}$. The set of all HF-instances of some database schema $\mathcal{S}$ is denoted by $\mathrm{HFinst}(\mathcal{S})$. If f is a permutation of $\mathbf{U}$ and I is a HF-instance, then $f(I)$ denotes the HF-instance obtained from I by applying f pointwise to all objects appearing in I, even if they appear within HF-sets.

For database schemas $\mathcal{S}_{\mathrm{in}}$ and $\mathcal{S}_{\mathrm{out}}$, an *HF-transformation* from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$ is a partial function $Q : \mathrm{inst}(\mathcal{S}_{\mathrm{in}}) \to \mathrm{HFinst}(\mathcal{S}_{\mathrm{out}})$ such that for each I for which $Q(I)$ is defined, we have

1. $|Q(I)| \subseteq |I| \cup \mathrm{HF}(|I|)$; and
2. for any permutation f of $\mathbf{U}$, also $Q(f(I))$ is defined, and equals $f(Q(I))$.

An ordinary instance I is said to be *isomorphic* to an HF-instance I' if there is a bijection f from $|I|$ to $|I'|$ such that $f(I) = I'$. Then a determinate transformation Q from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$ is said to be *isomorphic* to an HF-transformation Q' from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$, if Q' is defined precisely on all instances I for which there is an output instance J such that $Q(I, J)$, and all such J are isomorphic to $Q'(I)$.

We now recall the following connection between constructive transformations and HF-transformations:

Proposition 1 ([15]) *A determinate transformation is constructive if and only if it is isomorphic to some HF-transformation.*

Hence, in order to prove our theorem, it suffices to show that every tree-structured transformation is isomorphic to some HF-transformation. Thereto, let Q be a tree-structured transformation and let $Q(I, J)$. Note that J is tree-structured; we are going to define, for each vertex x of J, the *stamp* of x by bottom-up induction as follows. Let L be the set of labels of x. Now if x is a leaf, then the stamp of x is the ordered pair $(L, \emptyset)$. If x is not a leaf, we may assume by induction that the stamps for its children have already been defined; let C be the set of all those children's stamps. Then the stamp of x is the ordered pair (L, C).

We now observe that we can consider a stamp to be a HF-set with ur-elements in $|I|$. Indeed, ordered pairs, triples, and tuples, can be unambiguously represented by HF sets [12]. The same is true of natural numbers, so the numbers i that occur in labels can also be represented. Finally, by numbering the relation names of $\mathcal{S}_{\mathrm{out}}$, we can represent the relation names that occur in labels also by numbers. Now denote by J', the HF-instance obtained from J by replacing each vertex by its stamp, and define the HF-transformation

Q' by $Q'(I) := J'$. Since Q is determinate, Q' is well-defined, i.e., the definition of $Q'(I)$ does not depend on the chosen J. Moreover, Q' is a valid HF-transformation, by the genericity of Q. Furthermore, by definition, it is clear that Q is isomorphic to Q'. We thus have our desired HF-transformation and the theorem is proved.

References

[1] XQuery 1.0 and XPath 2.0 data model. W3C Working Draft, April 2005.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] S. Abiteboul and P.C. Kanellakis. Object identity as a query language primitive. *Journal of the ACM*, 45(5):798–842, 1998.

[4] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, 1990.

[5] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124, 1991.

[6] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Conference Record, 6th ACM Symposium on Principles of Programming Languages*, pages 110–120, 1979.

[7] M. Andries and J. Paredaens. On instance-completeness of database query languages involving object creation. *Journal of Computer and System Sciences*, 52(2):357–373, 1996.

[8] F. Bancilhon. On the completeness of query languages for relational data bases. In *Proceedings 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, 1978.

[9] J. Barwise. *Admissible Sets and Structures*. Springer-Verlag, 1975.

[10] A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[11] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), 1994.

[12] P. Halmos. *Naive Set Theory*. Van Nostrand Reinhold, 1960.

[13] R. Hull and C.K. Yap. The format model, a theory of database organization. *Journal of the ACM*, 31(3):518–537, 1984.

[14] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, 1978.

[15] J. Van den Bussche, D. Van Gucht, M. Andries, and M. Gyssens. On the completeness of object-creating database transformation languages. *Journal of the ACM*, 44(2):272–319, 1997.

[16] Extensible markup language (XML) 1.0 (second edition). W3C Recommendation 6 October 2000.

[17] XQuery 1.0: An XML query language. W3C Working Draft 20 December 2001.