The Use of Information Capacity in Schema Integration and Translation

R. J. Miller* Y. E. Ioannidis† R. Ramakrishnan‡

Dept. of Computer Sciences, Univ. of Wisconsin-Madison
{rmiller, yannis, raghu}@cs.wisc.edu

Abstract

In this paper, we carefully explore the assumptions behind using information capacity equivalence as a measure of correctness for judging transformed schemas in schema integration and translation methodologies. We present a classification of common integration and translation tasks based on their operational goals and derive from them the relative information capacity requirements of the original and transformed schemas. We show that for many tasks, information capacity equivalence of the schemas is not strictly required. Based on this, we present a new definition of correctness that reflects each undertaken task. We then examine existing methodologies and show how anomalies can arise when using those that do not meet the proposed correctness criteria.

1 Introduction

Formal work on schema equivalence has largely been ignored within practical schema integration and translation tools. Practitioners have felt that theoretical work is too narrow in scope to be applicable to the problems they face [RR89]. As a result their work is driven by an intuitive, rather than formal, notion of correctness. Some recent work on translation and integration has successfully used information capacity equivalence as a basis for judging the correctness of transformed schemas [Hul86, MS92, RR87, and others]. Such work formally provides sets of equivalence preserving transformations of schemas for specific data models.

We take a closer look at the assumptions behind using information capacity equivalence as a measure of correctness by examining a number of common tasks that require schema integration or translation. We present a classification of these tasks based on their operational goals and derive from them the information capacity requirements of the original and transformed schemas. For many tasks, information capacity equivalence of the schemas is not required. Rather it is sufficient to guarantee dominance of either the original or the transformed schemas. Based on this result, a new definition of correctness for transformed schemas is presented that takes into account the integration task and its operational goals.

We examine the literature on schema transformations with this new definition of correctness in mind. We identify many transformations, proposed within several different methodologies, that do not achieve their operational goals, and articulate the additional assumptions required to rectify the problem. As part of this process, we highlight the anomalies that can arise due to errors in the transformation rules and show how query mappings that are based on incorrect schema transformations can produce incomplete or inconsistent answers.

We then extend our definition of correctness to prohibit the generation of transformed schemas containing a conflict. We distinguish heterogeneity of schemas (structural or constraint mismatch) from conflicts created by constraints on schemas that cannot be simultaneously satisfied and show that determining if a transformed schema contains a conflict is undecidable in general. However, in many practical situations a tool can detect conflict and use this knowledge to correct errors in the specification of schemas.

We conclude by showing the value of this work to practitioners. The new definition of correctness lends itself to use in practical tools. Translation and integration tools must work with incomplete information. However, knowledge of the assumptions necessary to ensure correct

*Partially supported by NSF grant IRI-9157368.

†Partially supported by grants from NSF (IRI-9113736 and IRI-9157368 (PYI Award)), DEC, IBM, HP, and AT&T.

‡Partially supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering and by grants from NSF (IRI-9011563 and a PYI Award), DEC, Tandem, and Xerox.

Proceedings of the 19th VLDB Conference
Dublin, Ireland 1993

transformations can be included in a tool and used to intelligently ask a designer about additional constraints that may hold in the schemas to be integrated. Because correctness is based on information capacity, not preservation of opaque dependencies, missing information that must be inferred by the system can be formulated in terms familiar to the designer.

2 Information Capacity

Before introducing the notion of information capacity, we define some useful concepts that may be interpreted differently in different contexts.

Definition 2.1 Let A and B be sets. A mapping (binary relation) $f : A \rightarrow B$ is *functional* if for any $a \in A$ there exists at most one $b \in B$ such that $f(a) = b$; *injective* if its inverse is functional; *total* if it is defined on every element of A; and *surjective* (onto) if its inverse is total. A functional, injective, total, surjective mapping is a *bijection*.

Let $I(S)$ denote the set of all valid instances of some schema S. A schema S conveys information about the universe it models. This information is essentially captured by $I(S)$ which contains all the possible states of the modeled universe. From a different perspective, the *information capacity* of S determines its set of instances $I(S)$. Two schemas can be compared based on information capacity[1], where intuitively, a schema $S2$ has more information capacity than a schema $S1$ if every instance of $S1$ can be mapped to an instance of $S2$ without loss of information. Specifically, it must be possible to recover the original instance from its image under the mapping. The above are made precise below.

Definition 2.2 An *information capacity preserving mapping* (or information preserving mapping) between the instances of two schemas $S1$ and $S2$ is a total, injective function $f : I(S1) \rightarrow I(S2)$.

Definition 2.3 If $f : I(S1) \rightarrow I(S2)$ is an information capacity preserving mapping, then *S2 dominates S1 via* f, denoted $S1 \preceq S2$.

Definition 2.4 An *equivalence preserving mapping* between the instances of two schemas $S1$ and $S2$ is a bijection $f : I(S1) \rightarrow I(S2)$.

Definition 2.5 If $f : I(S1) \rightarrow I(S2)$ is an equivalence preserving mapping, then $S1$ and $S2$ are *equivalent* via f, denoted $S1 \equiv S2$.

Example 2.1 To illustrate the above, consider the simple relational schemas $S1$: emp1(*eno*, ename, sal) and $S2$: emp2(*eno*, ename, sal, age). The attribute *eno* is the key in both tables. One may define many intuitive information preserving mappings from $I(S1)$ to $I(S2)$, where each instance of emp1 is mapped to some instance of emp2 with the same employees (i.e., the same *eno*, but also the same ename and sal), ignoring the values of age. Therefore, $S2$ dominates $S1$ ($S1 \preceq S2$), which is to be expected, since more information is captured in emp2 than in emp1. □

In principle, arbitrary mappings f may be used to satisfy the above definitions of dominance and equivalence. In fact, the definitions do not even require that the mappings be finitely specifiable; they can simply be an infinite list of pairs of schema instances. Clearly, such mappings are of little use in a practical context. Various restricted classes of mappings have been proposed in the past [Hul86], for example, *internal* mappings, which only reorganize and do not invent values, and mappings that are queries in some query language. While such mappings have many desirable properties, many of them are still unacceptable in a practical environment, because schema instances with no intuitive relationship between them are allowed to be associated via the mappings.

In practice, the notions of dominance and equivalence are most useful if the associated mappings are required to capture a meaningful semantic correspondence between schemas. None of the classes of mappings discussed in the literature has been shown to guarantee this [Hul86]. In fact, existing systems operate at the schema level (using transformations between schema components) rather than at the instance level.

Definition 2.6 Let $\mathcal{S}1$ and $\mathcal{S}2$ be families (sets) of schemas. A *(schema) transformation* is a total function $F : \mathcal{S}1 \rightarrow \mathcal{S}2$. If $\mathcal{S}1$ and $\mathcal{S}2$ are specified in different data models, the transformation F is sometimes called a *translation*.

A schema transformations should induce a mapping between the sets of instances of the schemas. A transformation is desirable if it induces an equivalence preserving mapping.

Definition 2.7 A transformation $F : \mathcal{S}1 \rightarrow \mathcal{S}2$ is an *equivalence preserving transformation* if for all $S1 \in \mathcal{S}1$, $S1 \equiv F(S1)$.[2]

Schema transformations are not arbitrary functions, but are usually defined by a bijection (respectively a total, injective function) between components of the original and the transformed schema when schema equivalence (respectively dominance) is desired. Moreover, they are usually constrained to induce only internal mappings on instances. Since schema transformations are defined on finite schemas, they can be finitely specified, and so ideally can the instance mappings that they induce.

Our treatment of integration and translation in this paper is based entirely on schema transformations and on instance mappings that these induce. In the next section, we show that by ensuring that such mappings

[1] Information capacity has been used extensively in the translation and integration literature [AABM82, BC86, Bor78, Eic91, Hul86, HY84, MS92, Ris82, RR87, and others]. While the precise details of the definitions differ among authors, we present definitions that are in keeping with the spirit of the existing literature.

[2] Note that this condition implies the *data model commutative mapping principle* described elsewhere [Kal90] and corresponds to the definition of *lossless schema transformations* [Tro93].

are in fact information capacity preserving some very important operational goals can be achieved.

3 Taxonomy of Schema Integration and Translation Tasks

Schema translation and integration are necessary components of several diverse tasks with very different goals and operational requirements. Unfortunately, in many discussions of schema integration and translation, the specific task in mind is not made explicit, which results in occasional errors and misconceptions.

Our approach to developing correctness criteria for integration and translation consists of two steps. First, we identify a collection of operational goals that arise in the context of different integration tasks involving a pair of schemas. For each goal, we identify what relationship exists between the two schemas in terms of information capacity (Section 3.1). Second, we identify the operational goals implicit in a given integration task. We then use the analysis of goals presented earlier to determine what information capacity based constraints must hold with respect to schemas involved in the integration task (Sections 3.2, 3.3, and 3.4). In Section 5, we use the formal basis for correctness developed in this section and examine several transformations proposed in the literature in conjunction with different integration tasks.

3.1 Operational Goals and Relative Information Capacity

Let $S1$ and $S2$ be two schemas in some data model(s). Consider a system where $S1$ is used at the user-interface level and $S2$ is used at the database level. That is, users interact with the system through $S1$, while the data is stored under $S2$. We call such a system *unidirectional*. We identify three possible operational goals for unidirectional systems. In what follows, $i1 \in I(S1)$, $i2 \in I(S2)$, and $i1 = f(i2)$ for some mapping f being discussed.

(G1) Querying through $S1$ the data stored under $S2$. This is the minimum possible operational goal and captures the case where $S1$ is a view of $S2$ in the traditional sense. To achieve G1, any query on $S1$ must correspond to a unique query on $S2$ that returns the same answer. For that, it is sufficient for the view definition to induce at the instance level a total function $f : I(S2) \rightarrow I(S1)$. In that case, for a query q on $i1$, the following holds:

$$q(i1) = q(f(i2)) = q \circ f(i2). \quad (1)$$

That is, the query q on $i1$ is mapped to the unique query $q \circ f$ on $i2$. Based on the required properties of f (a total function), we conclude that f does not have to be information preserving to achieve G1: the information capacities of $S1$ and $S2$ may be incommensurate.

(G2) The goal G1, plus viewing through $S1$ the entire database stored under $S2$. To achieve G2, we need a total function $f : I(S2) \rightarrow I(S1)$ as above, but we also need f to not lose any information. An instance of $S1$ should uniquely determine an instance of $S2$, i.e., f should also be injective so that its inverse f^{-1} is well defined (albeit possibly not for all instances in $I(S1)$). Then, by the equalities of (1), one may use f^{-1} as a query on $i1$ to retrieve/view the entire database instance $i2$. More formally, $f^{-1}(i1) = f^{-1}(f(i2)) = i2$. Therefore, based on the required properties of f (a total injective function), we conclude that f must be an information preserving mapping to achieve G2: $S1$ must dominate $S2$.

(G3) The goal G2, plus updating through $S1$ the data stored under $S2$. To achieve G3, at a minimum we need a total injective function as above. Consider an update u that changes $i1$ to a new instance $i1'$, i.e., $u(i1) = i1'$. To perform the update on the underlying database, any $i1'$ must determine a unique instance $i2'$ of $S2$, i.e., f must also be surjective (onto $I(S1)$). In that case, because f is injective, f^{-1} is also uniquely defined on any $i1'$. Hence, $u(i1) = i1' \Leftrightarrow u(f(i2)) = f(i2') \Leftrightarrow f^{-1}(u(f(i2))) = f^{-1}(f(i2')) = i2'$ The composition $f^{-1} \circ u \circ f$ corresponds to the update on $S2$ that generates the unique new instance $i2'$. Therefore, based on the required properties of f (a bijection), we conclude that both f and f^{-1} must be information preserving mappings to achieve G3: $S2$ and $S1$ must be equivalent.[3]

In addition to the above, consider a system where some data is stored under $S1$ and some under $S2$. Moreover, some users interact with the system through $S1$ and some through $S2$. We call such a system *bidirectional*. As an example, two heterogeneous independent applications exchanging data in a distributed environment form a bidirectional system. We identify a unique operational goal for such a system.

(G4) Querying through $S1$ the data stored under $S2$ and vice-versa. For practical purposes, a single instance-level mapping is desired. Clearly, this goal is equivalent to G1 for both directions between $S1$ and $S2$. Therefore, f should be a total function (for one direction) and a surjective injection (for the other direction), i.e., a bijection. Therefore, to achieve G4 $S2$ and $S1$ must be equivalent. Given this, updates can be done through both $S1$ and $S2$.

Summarizing the above, we have identified several possible operational goals for systems that involve two schemas. For each one of these goals, we have

[3] We note that goals **G1-G3** also put implicit constraints on the query and update languages of S1 and S2. For **G1** (respectively **G2**), $g \circ f$ (respectively f^{-1}) must be a query on $S2$. For **G3**, $f^{-1} \circ u \circ f$ must be a valid update on $S2$. Such language constraints are discussed elsewhere [Kal90].

demonstrated whether or not the two schemas must be in a dominance or equivalence relation. This is extremely valuable in analyzing various integration and translation tasks in the following subsections. In addition, we have shown how such properties are necessary to be able to translate queries and updates between schemas automatically, an absolute necessity for practical heterogeneous systems.

3.2 Database Integration

Database integration or **global schema design** is a process that takes several, possibly heterogeneous, schemas and integrates them into a view that provides a uniform interface for all the schemas [ADD+91, BLN86, DH84, LNE89, MB81, ME84, et al]. If the local schemas are specified in different data models, they may be translated into a common model before integration is performed. The process is depicted in the top part of Figure 1, where both an integration and a translation step are shown.

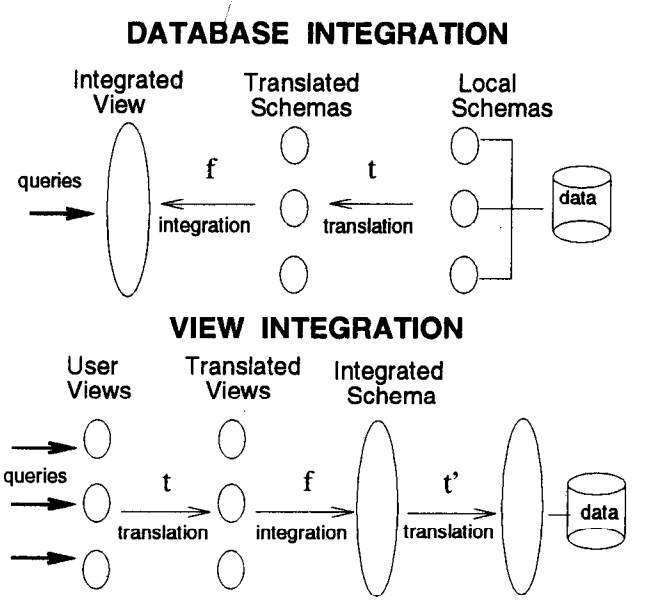

Figure 1: Information capacity requirements of tasks.

Clearly, database integration generates a unidirectional system, where $S1$ is the integrated view and $S2$ is the union of the local schemas. Given that this is an integration task, the view must faithfully represent the integration of all schemas. Hence, the operational goals of the system resulting from the integration may be G2 and possibly G3 if updates are allowed through the view. By the analysis in Section 3.1, in the former case the integrated view ($S1$) must dominate the union of the local schemas ($S2$), while in the latter case the two must be equivalent. The composition of the integration mapping f and the translation mapping t shown in Figure 1 is the information (equivalence) preserving mapping from $I(S2)$ to $I(S1)$, so both f and t must be information (equivalence) preserving as well.

3.3 View Integration

View integration or **logical database design** is a process that takes a set of user views and logically integrates them into a single conceptual schema [BLN86, BC86, SZ91, LNE89]. These views contain requirements for the portion of a database of interest to different users. The result of their integration is the schema for an actual database. The user views are optionally maintained to permit users to query the database using their own customized interface. If the views are specified in different data models they may be translated into a common model before the integration is done. Depending on the model used for integration, the resulting conceptual schema may also be translated into a target schema in a final step. For example, the ER model is commonly used for schema integration and the relation model used for the target schemas. The process is depicted in the bottom part of Figure 1, where both an integration and two optional translation steps are shown.

As in database integration, when the user views are maintained, view integration generates a unidirectional system. However, the roles of $S1$ and $S2$ are reversed: $S1$ corresponds to the collection of the user views and $S2$ is the integrated schema. The goals remain the same as before, G2 and possibly G3, and therefore the conclusions are reversed: for G2, the integrated schema ($S2$) must be dominated by the user views ($S1$), while for G3, the two must be equivalent. Analogously to the previous case, the inverses of the integration mapping f and the translation mappings t and t' (Figure 1) must be information preserving.

If the user views are not maintained and no interaction with the data is performed through them, then the resulting system is in some sense 'nondirectional', and does not have any operational goals of the type discussed in Section 3.1. Therefore, it may not be strictly necessary for the integrated schema to be dominated by the user views. However, by the nature of the view integration process, the integrated schema often contains additional constraints, including interschema constraints, that are not expressed on any of the individual views. Moreover, the constraints on the views capture useful information that must be preserved in the integrated schema as well. Therefore, it is again desirable for the integrated schema to be of less information capacity. [4]

[4] Informally, the constraints may be treated as information that must be preserved and perhaps enhanced during the design process. These constraints commonly restrict the information capacity of the views. In such cases, it may still be a desirable (though no longer strictly necessary) goal to preserve these constraints by preserving information capacity.

3.4 Schema Translation

As we saw earlier, *schema translation* is often combined with database integration [ADD+91] or view integration [RR87, MS92] in a heterogeneous environment. The information capacity requirements on the translated schemas depend on the task at hand. This difference is often not recognized in practice. Schema translation may also be required in situations not involving integration. In a unidirectional system, where both $S1$ and $S2$ are individual schemas, the mapping from $S2$ to $S1$ involves only a translation. Assuming that G2 is the most common goal for such systems, $S1$ should dominate $S2$ (where for database integration, $S1$ is the integrated view and for view integration $S1$ is the set of original user views). Also, in a bidirectional system, data is exchanged between the two parts of the system that must be translated. Having G4 as the goal, the two schemas must be equivalent.

3.5 Comments

In practice, in the process of any of the two types of integration tasks discussed above, designers may provide additional information that affects the information capacity of the original schemas in the opposite direction from the one described in Sections 3.2 and 3.3. For database integration, the reason is that the original schemas are often incomplete because they are specified in a model that does not support the expression of all constraints that hold in the universe of discourse. Additionally, there may be interschema constraints that hold in the integrated view but are not specified in any one of the original schemas. For view integration, the reason is that it may be revealed during integration that the constraints on the views are not valid in the integrated schema. In both cases, such information may lead to an integrated schema that is not in the appropriate dominance relation with the original schemas as described above. However, the additional information provided by the designer may be considered as part of the original schemas; the fact that the designer is the source is irrelevant. Taking this into account, the dominance relations are as described. We discuss how such missing or inconsistent constraints may be taken into account in Section 7.

4 The SIG Formalism

In Section 5, we study several methodologies that have been proposed for view integration or database integration in light of the information capacity requirements that we have described above. The methodologies that we consider use various data models to express the transformations. To ease the task of analyzing and presenting the transformations, we use a single data model for describing the relevant schema families. Specifically, we employ *schema intension graphs* (SIGs), a formalism defined elsewhere [MIR]. Within SIGs, constraints that impact the information capacity of a schema are explicitly represented in a simple graph notation. These graphs are therefore a convenient tool for proving or disproving equivalence of schemas.[5]

We define informally the features of SIGs that are relevant to our subsequent discussion. A *schema intension graph* is a graph $G = (N, E)$ defined by two finite sets N and E. The set N contains (i) a typed set of symbols each used to represent a finite set of data values, and (ii) arbitrary products and sums of these symbols used to denote cross products and unions of the corresponding sets, respectively.

The set E contains labeled edges between two nodes. Edges are used to represent binary relations on the sets assigned to the nodes. An edge e is denoted by $e : X - Y$ indicating that it is an edge between nodes X and Y. The set E may contain edges between product and sum nodes as well as multiple edges between the same pair of nodes. Some of the edges of E may be designated as *projection* or *selection* edges. A selection edge, denoted $\sigma : X - Y$, indicates that the set assigned to the node X is a subset of the set assigned to Y. The edge σ represents the identity relation on the elements of X. A projection edge, denoted $\pi^X_{X\times Y} : X \times Y - X$ or $\pi^Y_{X\times Y} : X \times Y - Y$, represents the projection on X (respectively Y) values.

An *instance* $\Im$ of a schema intension graph G (i.e., a database state) is a function defined on the sets N and E that assigns specific sets to nodes (denoted $\Im[X]$ for $X \in N$) and specific binary relations to edges (denoted $\Im[e]$ for $e \in E$). The set of all instances of G is $I(G) = \{\Im \mid \Im \text{ is an instance of } G\}$.

Each edge of a SIG is annotated with a (possibly empty) set of properties. Each property is a constraint that restricts the valid set of binary relations that may be assigned to an edge by an instance. Specifically, an annotation of a SIG $G = (N, E)$ is a function $\mathcal{A}$ whose domain is the set of edges E. For all $e \in E$, $\mathcal{A}(e) \subseteq \{f, i, s, t\}$. An instance $\Im$ of G is a *valid instance* of $\mathcal{A}$ if for all $e \in E$, if $f \in \mathcal{A}(e)$ (respectively i, s or $t \in \mathcal{A}(e)$) then $\Im[e]$ is a functional (respectively injective, surjective or total) binary relation.

In reasoning about SIGs, edges may be combined using two constructors $\ll, \gg$ and $[[,]]$.

- Two edges $e_1 : Z - X, e_2 : Z - Y$ can be combined into the edge $\ll e_1, e_2 \gg : Z - (X \times Y)$ called the constructed product of e_1 and e_2. In any instance of the constructed edge, $(z, (x, y)) \in \Im[\ll e_1, e_2 \gg]$ iff $(z, x) \in \Im[e_1]$ and $(z, y) \in \Im[e_2]$.

[5] We are not recommending that practitioners take a step backward in time and start using a simpler rather than more expressive data model for integration. We use SIGs here as an expository tool. We also stress that SIGs are in fact powerful enough to express the relational schemas with functional and inclusion dependencies as well as schemas expressed in models with inheritance [MIR].

- Two edges $e_1 : X-Z, e_2 : Y-Z$ can be combined into the edge $[[e_1, e_2]] : (X+Y)-Z$ called the constructed sum of e_1 and e_2. In any instance of the constructed edge $(w, z) \in \Im[\ [[e_1, e_2]]\ ]$ iff $(w, z) \in \Im[e_1]$ or $(w, z) \in \Im[e_2]$.

5 Analysis of Transformations

Using the SIG formalism, we have examined a large number of database or view integration methodologies that have been proposed in the literature. Based on the stated operational goals of each methodology, we have used the taxonomy of Section 3 to study the transformations proposed within the methodology and understand their properties with respect to information capacity. While in most cases the notion of correctness is not explicitly stated in the original references, based on intuitive assumptions, we have been able to classify most of the previous work. The entire effort has been very beneficial. First, transformation rules originally described in the context of very diverse models, have been unified under the SIG formalism. This has permitted the identification of common themes underlying many transformations. Second, the analysis of transformations based on information capacity has identified many errors in the existing literature. Several transformations have been found to not preserve information capacity, implying that any methodology incorporating them would fail to achieve its stated operational goals. Third, for each incorrect transformation, we have used information capacity arguments to identify the missing assumptions that would validate the transformation. Fourth, we have obtained a better understanding of how queries can be transformed between schemas using the instance-level mappings induced by the transformations that preserve information.

All these points have convinced us of the importance of using information capacity preservation as a correctness criterion for integration and translation. We illustrate the above by focusing on three broad classes of transformations. For each class, we have chosen one or two transformations from different methodologies among those studied and present the details of our analysis. The specific transformations were chosen because they have similar counterparts in other methodologies. For each transformation, we discuss (1) the methodology in which it is proposed; (2) our formulation of the transformation in SIGs; (3) a natural instance mapping induced by the transformation; (4) under what assumptions the mapping meets the goals of Section 3.1; (5) how the transformation may be correctly used in an integration task; (6) how queries can be translated based on the transformation; and (7) related transformations. In all cases, we only present the essential properties of the transformation and omit details that are not pertinent to our analysis.

5.1 Merging through Generalization

For both view and database integration, several methodologies have been proposed that begin with two or more schemas, union the schemas into a single schema, and perform restructuring (or merging) transformations on components of the new schema. The restructuring transformations are designed to identify common structures which are then combined to produce a conceptually cleaner schema. One of the most common types of transformations merges two or more object classes with overlapping attributes using generalization. In the following subsections, we analyze one such transformation from a specific methodology.

5.1.1 Description of Methodology

The specific transformation that we consider belongs to a database integration methodology that uses a simple semantic model containing classes and two types of binary relationships, corresponding to attribute relationships (key and non-key) and generalization [MB81, Mot87]. The stated goal of the work is to provide a faithful representation of the underlying schemas and to allow queries on the integrated view (referred to as a *superview*) to be automatically transformed into queries against the component schemas. This is goal G2 which requires that the integrated schema dominate the original schemas.

5.1.2 Description of Transformation

The *meet* transformation restructures a schema containing two classes with a common key. The two classes typically come from different schemas that are integrated. In the transformed schema, all common attributes are removed from the original classes and a superclass with these attributes is created.

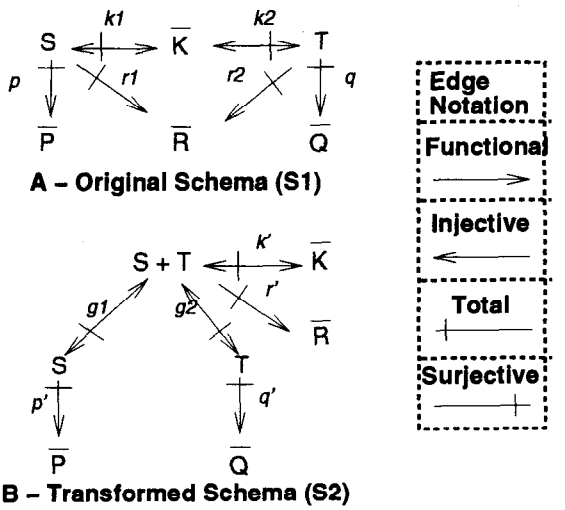

Figure 2: The meet transformation.

Let S and T be classes with a common key composed of the attributes $\{K_1, K_2, ..., K_k\} = K$. Let R =

$\{R_1, R_2, ..., R_r\}$ be the set of all common non-key attributes. Additionally, S has non-key attributes $P = \{P_1, P_2, ..., P_p\}$ and T has non-key attributes $Q = \{Q_1, Q_2, ..., Q_q\}$. The SIG representing this schema is shown in Figure 2A (Schema S1).[6] For any set of nodes N, the node $\bar{N}$ represents the product of all nodes in N. In the SIG representation, we typically do not depict all projection edges. Intuitively, an attribute relationship from S to P_i is represented by the composition of the edge p and the projection edge $\pi_P^{P_i}$.

The meet transformation converts S1 into S2, shown in Figure 2B. Schema S2 contains a new class $S + T$ representing the generalization of S and T. All common attributes are made attributes of $S+T$. The edges $g1$ and $g2$ represent generalization relationships. The classes S and T "inherit" all attributes in K and R via the composition of the generalization edges with the edges k' and r'.

5.1.3 Induced Mapping on Instances

While instance level mappings are not explicitly defined in the discussion of the meet transformation, the authors suggest the following mapping. Let $f : I(S1) \rightarrow I(S2)$ and $\Im \in I(S1)$.

- The new node $S + T$ is populated with the union of elements in S and T, $f(\Im)[S + T] = \Im[S] \cup \Im[T]$.
- For all other nodes X in $S2$, $f(\Im)[X] = \Im[X]$.
- The edges are populated as follows:
  $f(\Im)[p'] = \Im[p]$; $f(\Im)[q'] = \Im[q]$;
  $f(\Im)[r'] = \Im[\ [[r1, r2]]\ ]$; $f(\Im)[k'] = \Im[\ [[k1, k2]]\ ]$;
  $f(\Im)[g1] = id_{\Im[S]}$; $f(\Im)[g2] = id_{\Im[T]}$;
  $f(\Im)[\pi_X^Y] = \Im[\pi_X^Y]$.

To be correct, the transformation must induce a mapping that is a total function. Therefore, the authors state that if an element of S and an element of T have the same key then they must agree on all attributes of $\bar{R}$. Otherwise, the relation $\Im[\ [[r1, r2]]\ ]$ (which determines the nonkey attributes of the merged class) would not necessarily be functional, and not every valid instance of $S1$ could be mapped to a unique valid instance of $S2$.

5.1.4 Additional Requirements for Information Capacity Preservation

Even under the above constraint, the meet transformation is not information preserving, i.e., $S2$ does not dominate $S1$. The relation $\Im[\ [[k1, k2]]\ ]$ is not guaranteed to be functional or injective. Hence, it does not always form a valid instance of the edge k'. To illustrate this, consider schemas S1 and S2 of Figure 3 which conform to the structure of the abstract schemas in Figure 2. According to the constraints on S1, two different people, an instructor i and a student s, may both be named "A. Walker". The edge *Name* in the integrated schema, which is populated with $[[Std-name, Inst-name]]$, would therefore contain the pairs (s, "A. Walker") and (i, "A. Walker"). So the *Name* edge is not injective. Conversely, if the same person p is an instructor and a student, he or she may have *Std-name* "A. Walker" and *Inst-name* "Dr. Walker". The edge *Name* would then contain the pairs (p, "A. Walker") and (p, "Dr. Walker"). So *Name* is not guaranteed to be functional.

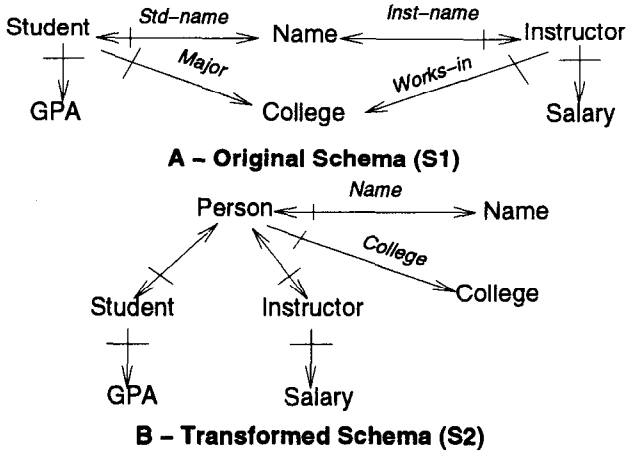

Figure 3: The meet transformation on a specific schema.

The additional requirement necessary to ensure $S1 \preceq S2$ is that two elements in $\Im[S]$ and $\Im[T]$ share the same key if and only if they are identical. Formally, if $s \in \Im[S]$ and $t \in \Im[T]$ then $s = t$ if and only if for a unique k $(s, k) \in \Im[k1]$ and $(t, k) \in \Im[k2]$. Keys must be unique not only within a node but across nodes that are to be merged.

If the additional constraint on uniqueness of keys is imposed on instances of $S1$ then f is an information preserving mapping. This mapping is not surjective, however, since a valid instance of $S2$ may populate the edges $g1$ and $g2$ with relations other than the identity relation. These instances are not in the range of f. In fact, no information preserving mapping from $S2$ to $S1$ exists and so $S2 \not\preceq S1$. Intuitively, in any mapping of instances of $S2$ to instances of $S1$, the two binary relations $g1$ and $g2$ are lost.

5.1.5 Usage

Before this transformation is used in a database integration methodology it should be verified that the constraint on keys holds across the component schemas. The existence of the mapping f confirms that the goal G2 can be met for all instances meeting this constraint. This transformation is not appropriate for view integration methodologies (even if the uniqueness constraint holds) since the integrated schema has strictly more information capacity. The transformation could be modified to

[6] In this figure and later figures, we represent information from the two original schemas together in the schema $S1$. The transformed schema is $S2$.

produce a schema in which the edges $g1$ and $g2$ are restricted to be selection edges (that is edges that must be populated with the identity relation). It is a straightforward exercise to show that such a schema is dominated by the original schema. Equivalence would hold for this new transformation if the key uniqueness constraint holds in the original schema. With this change, goal $G3$ is achieved.

5.1.6 Mapping Queries

We have shown that the information preserving mapping f is necessary to ensure that the G2 operational goal can be achieved. We now demonstrate how this is done in practice by giving an example of transforming queries on the superview $S2$ to queries on the underlying schema $S1$. Below is a sample query on the schema S2 of Figure 3.[7]

Range of P **is** Person (q1)
Retrieve P **where** Name(P) = "A. Walker"

This query must be translated into a query on the underlying schema S1. The schema transformation can be used to automatically generate the set of queries $q2$.

Range of P1 **is** Student (q2)
Range of P2 **is** Instructor
Retrieve P1 **where** Std-name(P1) = "A. Walker"
union
Retrieve P2 **where** Inst-name(P1) = "A. Walker"

The two queries in q2 are the result of composing q1 with the mapping f. The node *Person* is mapped by f to the union of *Student* and *Instructor* and so the query q1 is replaced by two queries. The edge *Name* maps to the constructed sum of the edges *Std-name* and *Inst-name*. Hence, in each of the two transformed queries *Name* is replaced by *Std-name* and *Inst-name*, respectively. The queries of q2 produce the same result as q1 on any potential instance of the database, i.e., $q2(i2) = q2 \circ f(i1) = q1(i1)$.

Consider what would happen if the constraint on uniqueness of keys across Student and Instructor instances did not hold. Let $\Im$ be an instance of $S1$ containing a student and an employee both named *A. Walker*. Each person is unique and therefore has a different identifier stored in the *Student* and *Instructor* nodes. In $S2$, *Name* is a key for people so the user is expecting query q1 to return a single person. Query q2, however, will return both people. In fact, the response to any query on $S1$ must either omit information contained in $\Im$ about one of these two people or violate the constraints of the schema $S2$. This simple example demonstrates the importance of verifying the correctness of a transformation.

[7] We use an intuitive QUEL-like query language to illustrate query modification.

5.1.7 Related Work

Other methodologies describe related transformations by example or provide mechanisms for defining views with similar effects [DH84, Eic91]. Similar assumptions are typically required to ensure these transformations create equivalent schemas. This same example can also be modified to show that the *information ordering* used in other schema merging proposals does not correspond to information capacity dominance [BDK92].

5.2 Merging Object Classes

An alternative strategy to unioning schemas and then performing restructuring transformations is to superimpose structures that have been identified as referring to the same set of real world concepts. We examine one such transformation in the following subsections.

5.2.1 Description of Methodology

The specific methodology we consider is designed to be used in both view integration and database integration [NEL86, LNE89]. To achieve this goal, the transformed schema must have equivalent information capacity to the original schemas (Section 3). The methodology uses the entity-category-relationship (ECR) model, an extension to the ER model [EHW85].

5.2.2 Description of Transformation

We describe a *merge* transformation for entity classes with a single attribute as key. We present only one alternative of a family of transformations from this methodology [LNE89, NEL86]. Consider two schemas containing two entity classes with key nodes K_1 and K_2 respectively. Both entity classes have common attributes $S = \{S_1, ..., S_S\}$. Additionally, K_1 has attributes $A = \{A_1, ..., A_A\}$ and K_2 has attributes $B = \{B_1, ..., B_B\}$. Suppose these two classes have been identified as referring to the same real world concept, so there is a bijective correspondence between instances of the two classes. This scenario is represented by the schema $S1$ of Figure 4A representing the union of the two schemas with the bijective correspondence explicitly represented by the edge e. This schema is transformed into the integrated schema $S2$ of Figure 4B containing the single class K_1 which is associated with the union of the two attribute sets.

5.2.3 Induced Mapping on Instances

While the authors do not discuss an instance level mapping, an intuitive mapping $f : I(S1) \rightarrow I(S2)$ can be defined as follows. Let $\Im \in I(S1)$.

- For all nodes X in $S2$, $f(\Im)[X] = \Im[X]$.
- For the edge $f3$, $f(\Im)[f3] = \Im[\ll f1, (\pi 4 \circ f2 \circ e) \gg]$.
- Projection edges are mapped to the corresponding projection edges of S1. When there are two possible edges in S1 (resulting from the multiple paths to the nodes of $\bar{S}$) projections involving $\pi 2$ are used.

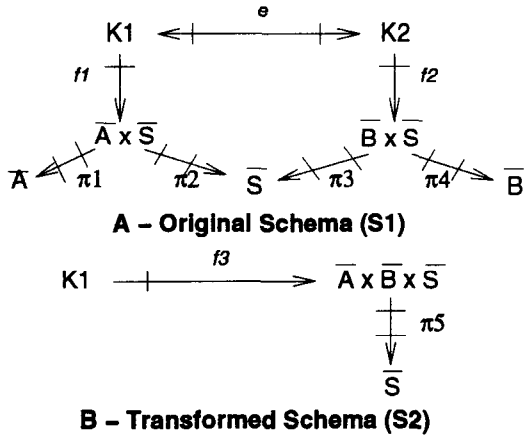

Figure 4: The result of merging two entities.

The combination of two total functional edges with the $\ll, \gg$ constructor is necessarily a total function, so $f3$ is only populated with total functions. The map f is therefore a total function mapping every valid instance of $S1$ to a unique valid instance of $S2$.

5.2.4 Additional Requirements for Information Capacity Preservation

The map f is clearly not injective since it loses information about the node $K2$ (and edge $\pi 3$). Any two instances of $S1$ that are identical except for values assigned to the node $K2$ will map to the same instance of $S2$. The following assumptions are sufficient for f to be an information preserving mapping. Since $S2$ does not represent values of $K2$ or the bijection e, $K2$ can be constrained to be identical to $K1$ (that is $K2$ and $K1$ must be assigned identical values in any valid instance) and e constrained to the identity map. Furthermore, in $S1$, there are two distinct paths representing binary relations between $K1$ and $\bar{S}$. If these two paths are constrained to be identical then an instance of the two paths can be uniquely represented by the single path from $K1$ to $\bar{S}$ in $S2$. This second constraint corresponds to the relationship uniqueness assumption (RUA), often asserted in relational theory, that a schema represents at most one relationship between any given set of attributes [AP82]. Under this assumption, and the assumption that the keys of the object classes are identical, the map f is an information preserving mapping and $S1 \preceq S2$.

Now consider the inverse mapping function $g : I(S2) \rightarrow I(S1)$ defined on instances $\Im \in I(S2)$.

- For all nodes $X \neq K2$ in $S1$, $g(\Im)[X] = \Im[X]$ and $g(\Im)[K2] = \Im[K1]$;
- $g(\Im)[e] = id_{\Im[K1]}$; $g(\Im)[f1] = \Im[\pi^{\bar{A}\times\bar{S}} \circ f3]$; $g(\Im)[f2] = \Im[\pi^{\bar{B}\times\bar{S}} \circ f3]$.

The map g is an information preserving mapping that carries every instance of $S2$ to a unique instance of $S1$ without losing any information. Hence, $S2 \preceq S1$.

5.2.5 Usage

The information preserving mapping g carries instances of the integrated schema to instances of the original schema and does not require any constraints or assumptions on the original schemas. Hence, g is sufficient to guarantee that the goal G2 can be achieved for view integration. However, instances of the original schema must satisfy the RUA and the relationship between key values restricted to the identity in order for f to be an information preserving transformation. Under these constraints, f is sufficient to guarantee that the goal G2 can be achieved for database integration. Furthermore, under these same constraints $g = f^{-1}$ and both f and g are equivalence preserving. Hence, $S1 \equiv S2$ and goal G3 can be met.

5.2.6 Mapping Queries

We consider an example in which the merge transformation is used for database integration so that queries on the integrated view $S2$ are translated into queries on the underlying schemas. Consider schemas S1 and S2 of Figure 5, which conform to the structure of the abstract schemas in Figure 4. Schema S1 contains two sets of nodes describing student records; the first is from the Registrar's view and the second from the Bursar's view. The edge e represents an interschema constraint not present in either user view.

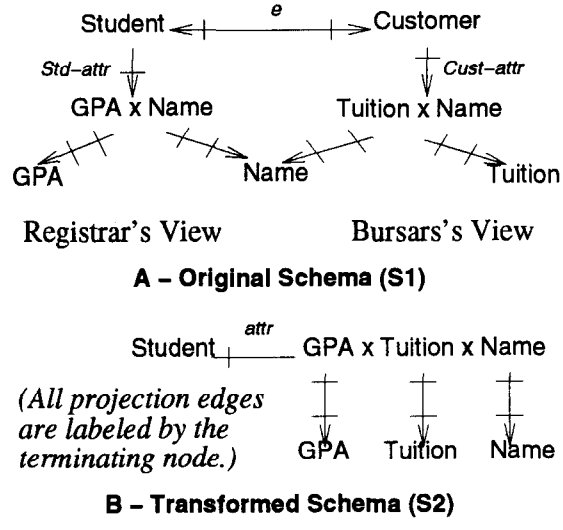

Figure 5: The merge transformation on a specific schema.

Consider the following query on the integrated view.

Range of C **is** Student (q1)
Retrieve C **where** Name(attr(C)) = "A. Walker"

This query must be translated into a query on the

underlying schemas. The mapping f can be used to automatically generate the query $q2$.

Range of C **is** Student (q2)
Retrieve C **where** Name(Std-attr(C)) = "A. Walker"

The node *Student* in $S2$ is mapped by f to the *Student* node in $S1$. The path $Name \circ attr$ is mapped to the path $Name \circ Std - attr$ (the mapping takes advantage of the definition of the $\ll, \gg$ constructor).

The mapping f always transforms a query about names into a query on the *Name* edge of the Registrar's schema. There is no way to query the *Name* edge of the Bursar's schema through the integrated view $S2$. If the same student has a different name within the Registrar's schema and the Bursar's schema then the view does not faithfully represent the full integration of the two underlying schemas. The reason for this is that the mapping f is information capacity preserving only under the assumption that names of identical people in the two schemas are identical.

5.2.7 Related Work

Additional merging transformations are suggested under the same proposal. Again, most of the transformations add constraints and so guarantee that the integrated schema is dominated by the original schemas. If the original schemas are assumed to satisfy the RUA then many of the transformations also guarantee dominance in the opposite direction.

Similar transformations are included in methodologies based on the relational model to merge two relations [BC86]. In that work, the precondition for applying the transformation is the existence of an integration constraint asserting that the projection of the two relations on the common attributes (including the key) is identical in all instances. This constraint is sufficient to ensure that the transformed schema has equivalent information capacity to the original. Other relational work has used less restrictive preconditions for merging two relations [MS92]. Null dependencies and general inclusion dependencies are used to ensure information capacity is preserved.

5.3 Structural Mismatch

The merging of schemas (whether via unioning or superimposition) is often preceded by a "conflict" resolution phase. During this phase similar information that is represented in different constructs within different schemas is identified and the mismatch is resolved by changing one of the structures. Methodologies based on the ER model typically have transformations to change attributes to entities (and vice versa) and entities to relationships (and vice versa) [BL84]. Methodologies using richer models including specialization or generalization have a greater array of transformations to handle the greater number of possible mismatches [SZ91, WE79].

5.3.1 Description of Methodology

We focus on a view integration methodology included in a database design tool [SZ91]. The tool recognizes structural differences and either applies a single resolution transformation or provides the designer with a choice of resolution transformations. We examine two situations and show how considering information capacity can 1) suggest an alternative transformation that may be appropriate under certain conditions and 2) guide the designer in the choice of transformations to apply. The proposal that we consider uses the Binary-Relationship (BR) model which models objects and binary relationships between objects.

5.3.2 Description of Transformation 1 - Entity-Relationship Mismatch

The first structural mismatch transformation concerns an entity-relationship (object-relationship) mismatch. An object class $O1$ in one schema is identified as having the same meaning as a relationship $r1$ in another (i.e., there is a bijective correspondence between instances of $O1$ and instances of $r1$). We consider the case where the relationship is n:1 as depicted in Figure 6A. According to the described integration methodology, the mismatch is resolved by adding a 1:1 relationship between $O1$ and the determining object class of the relationship. The new schema ($S2$) is depicted in Figure 6B.

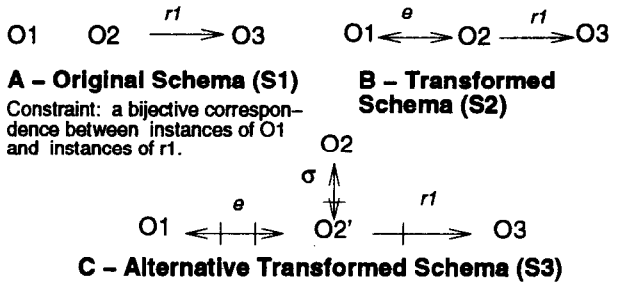

Figure 6: An entity and relationship that refer to the same concept.

5.3.3 Additional Requirements for Information Capacity Preservation

Schema $S2$ allows instances that do not correspond to instances of the original schema. Specifically, let $O1$ be the class Research-Assistantship (RAship) containing attributes for the salary, semester and job description of an RA. Let $r1$ be the relationship RAship, $O2$ the class of graduate students, and $O3$ the class of professors. So $r1$ represents student-professor pairs containing students that hold an RAship with a specific professor. A valid instance of $S2$ could associate an RA salary with a graduate student (via the edge e) who is not employed as an RA by any professor (via the edge $r1$). Such an instance violates the constraint in $S1$ that every instance

of the RAship class $O1$ must correspond to a unique instance of the RAship relationship $r1$ (and vice versa). So $S2$ has more information capacity than $S1$. This transformation therefore may not be appropriate for view integration.

Clearly, every instance of $S1$ could be mapped to an instance of $S2$ via an information preserving mapping. So $S1 \preceq S2$ and this transformation is appropriate to meet the goal G2 in a database integration context. However, G3 cannot be achieved.

To ensure that only graduate students participating in the relationship $r1$ (and therefore doing work for a specific professor) can be associated with an instance of $O1$ (and earn a salary), the set of graduate students holding RAs must be explicitly represented as in Figure 6C. There is a natural bijective mapping function between instances of $S1$ and $S3$ and so $S1 \equiv S3$ and this modified transformation may be used in view integration and also to achieve goal G3.

5.3.4 Description of Transformation 2 - Relationship-Generalization Mismatch

The next transformation involves a mismatch between a relationship edge and a generalization edge, both defined on the same two object classes. The two edges have been identified as referring to the same concept. Figure 7 shows the two constructs with one possible set of annotations of the relationship edge. To resolve this mismatch, the designer is given the option of changing either the relationship edge to a generalization edge or vice versa [SZ91]. The two constructs are then merged (superimposed). However, an analysis of information capacity can be used to guide the designer in making a choice.

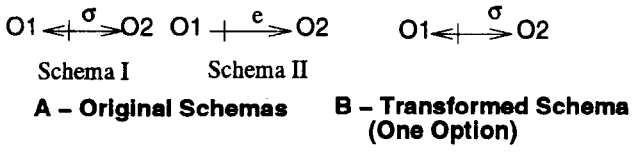

Figure 7: A generalization edge and a relationship edge that refer to the same concept.

5.3.5 Additional Requirements for Information Capacity Preservation

If the annotations on e are consistent with the generalization edge ($\mathcal{A}(e) \subseteq \{f, i, t\}$) then Schema II dominates Schema I. Schema I does not dominate Schema II since instances of the edge σ must be subsets of the identity. If $\mathcal{A}(e) = \{f, i, s, t\}$ the two schemas are incomparable in terms of information capacity. Not every bijective relation is a subset of the identity and not every subset of the identity is a bijection. Constraints must therefore be added or deleted to arrive at a schema that dominates (or is dominated by) both schemas.

5.3.6 Comments

There is a subtle distinction between the assumptions behind the resolution of this mismatch and the resolution of entity-relationship mismatches. The assertion that an entity and relationship refer to the same real world concept was interpreted as the existence of a bijection between instances of the two constructs. In the transformed schema this bijection is explicitly represented. In resolving mismatches between two types of relationships, however, the same assertion was interpreted as stating that instances of the two constructs are identical. This distinction is important in considering information capacity. In the first case, the designer is asserting that there is a new correspondence between values that is not captured in the original schemas and should be added in the integrated schema. In the second case, the designer is asserting the identity of values in two schemas to be integrated. This distinction is often glossed over in integration strategies and a designer may not realize that the same statement is being interpreted in different ways.

6 Conflicts

We have shown that schema translation and integration may require putting additional constraints on a schema or set of schemas. Any time the information capacity of the original schemas is reduced (even by the addition of interschema constraints) the possibility of an irreconcilable conflict in the schemas must be considered. As mentioned above, the word conflict has traditionally been used within the integration literature to indicate a mismatch or heterogeneity in the original schemas. For instance, a structural mismatch, where the same information is represented by different constructs of the data model, is often called a structural conflict. However, it is important to recognize the distinction between heterogeneity and true conflict. We reserve the term conflict to mean constraints in a schema or set of schemas that cannot be simultaneously satisfied.[8]

Information capacity can be used to formally understand and reason about conflicts. A conflict forces the information capacity of a schema or part of a schema to be the empty set.[9] The potential for conflict will depend on the language used to express constraints. In some data models, including the relational model with just functional dependencies, conflicts may not be possible. In more expressive formalisms, including schema intension graphs, conflicts may arise.

[8] We also note that conflict applies to a schema rather than to an instance of a schema that may hold contradictory information. Instance level conflicts are beyond the scope of this paper.

[9] Other authors refer to such schemas as *inconsistent* [AP86].

It is critical that a schema transformation tool recognize potential conflicts and use this information to guide the designer in correctly specifying constraints and choosing transformations. A tool that blindly takes the union of constraints specified on constructs may produce a schema for which certain constructs have no information capacity. These constructs could be removed from the schema without changing the information capacity of the schema. Clearly, such a situation could be very confusing. Furthermore, the burden should not be placed on the designer to recognize such situations.

6.1 An Example of Conflict

We consider an example of a conflict that may arise in view integration and discuss how a tool can guide the designer in resolving the conflict. Views I and II in Figure 8A represent two user views of information about students and workstation allocation in a university. From the CS Department's point of view, all students are either teaching assistants (TAs) or research assistants (RAs) and not both (so the node students is the disjoint sum of TAs and RAs). The department has the resources to dedicate *at least* one workstation to every student. The Bursar's office controls the grant money used to pay for resources allocated to RAs. Each grant assigned to a student may be used to pay for only one workstation. Every RA is assigned a workstation under at least one grant and furthermore, every workstation paid for by grant money must be associated with some student. These two different views are depicted in Figure 8A.

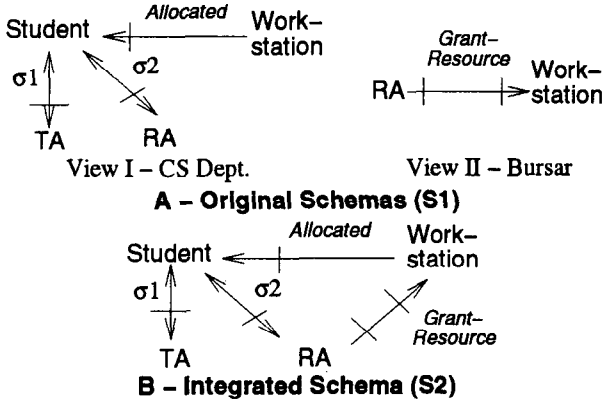

Figure 8: Integration of schemas via superimposition. Integrated schema has a conflict.

Suppose the designer has indicated that the node *Workstation* in View I is identical to the node *Workstation* in View II. An integration methodology that recognizes *Allocated* and *Grant-Resource* as distinct relationships and superimposes user views would create Schema S2. The constraints on instances imposed by the edges *Allocated*, *Grant-Resource* and $\sigma 2$ imply that for any valid instance $\Im$, $|\Im[Student]| \leq |\Im[Workstation]| \leq |\Im[RA]| \leq |\Im[Student]|$ (a simple inference rule can be used to deduce this). Hence, in any valid instance of the schema there are exactly as many students as RAs. As a result, there can never be any TAs (since the sets TAs and RAs are defined to be disjoint). It is unlikely that this is the intent of the schemas. Rather, it is probably the case that the set of workstations in View II is a subset of the workstations in View I.

It should not be up to the designer to recognize that this schema contains a conflict. Rather a tool should identify the conflict and guide the designer in resolving the conflict. Appropriate questions include: "Are there no TAs?" or "Is it always the case that there is a one-to-one correspondence between every RA and every workstation?" These questions can be used to determine that the edge *Grant-Resource* should not be surjective onto the Workstation node in the integrated schema.

6.2 Conflicts in SIGs

For SIGs, we can precisely define the notion of conflict. SIGs contain only sets and binary relations on sets. A conflict can occur if there are no valid instances for a node or edge due to the constraints contained in a specific graph.

Definition 6.1 For a SIG $G = (N, E)$ with annotation function $\mathcal{A}$, a node $X \in N$ is a *conflict node* if for all valid instances $\Im$, $\Im[X] = \emptyset$. An edge $e \in E$ of G is a *conflict edge* if for all valid instances $\Im$, $\Im[e] = \emptyset$.

An edge is a conflict edge if one of the incident nodes is a conflict node. So for SIGS we characterize conflicts by the existence of a node that is constrained to have no members in any valid instance. Ideally, a tool would be able to test for the presence of conflicts in an arbitrary schema. However, testing for conflicts (even in SIGs without selection, projection, or constructed edges) is undecidable in general.

Theorem 6.1 Let $G = (N, E)$ be a SIG with annotation function $\mathcal{A}$. Testing whether a node $X \in N$ is a conflict node is undecidable. [10]

The proof of this theorem uses the fact that annotations on edges can be used to express complex cardinality constraints on the size of nodes in any valid instance. Similar cardinality constraints can be expressed in most data models so the result generalizes. Schema III of Figure 8 expresses the equation $TA + RA \leq RA$ where the variable TA represents the size of the teaching assistant node and the variable RA represents the size of the research assistant node. Clearly, the only solution to this equation (in positive integers) is $TA = 0$. While the problem of deciding if an arbitrary polynomial has nonzero (integer) solutions is undecidable, the constraints arising in practice are likely to be such that

[10] The proof is contained in the full version of this paper.

simple algebraic techniques can be used to detect conflicts in many of these cases. In particular, schemas will often express only linear systems of equations (as in the above example).

7 Practical Tools and Information Capacity

In earlier sections, we argued that understanding schema transformations in terms of information capacity offered significant benefits in terms of i) identifying all relevant assumptions clearly, and ii) understanding the appropriateness of a transformation for a given integration/translation task. In this section, we address the question of what role such formally justified transformations have in a practical tool. Our discussion is brief, but we hope to convince the reader of two points. First, formally justified transformations offer some significant benefits. Second, they continue to be of value even when used in conjunction with other, non-rigorously justified, transformations and integration steps, as is to be expected in a practical setting for integration.

7.1 Outline of a Tool

The scope of an integration tool clearly goes beyond just schema transformations. For example, a tool must provide book-keeping support for several kinds of domain and catalog information, guidance to the user in identifying semantic matches between different elements of schemas to be integrated, and capabilities for graphical viewing of schemas. However, support for schema transformations is a very important aspect of such a tool, with great potential for automation, and it is this aspect that we consider here. A tool must contain a catalog of transformations for the data model(s) of interest. For each transformation, some information is maintained, for example: preconditions on the applicability of the transformation (e.g., there must be two classes that share a key and possibly some nonkey attributes), the induced instance mapping, and its properties.

A set of transformations can be used in several ways. A user could choose to apply a given transformation on a given pair of input schemas (or fragments of schemas). Alternatively, after the user specifies a certain amount of information, the system automatically chooses transformations and schema fragments on which to apply them.

7.2 Benefits of Including Support for Schema Transformations

Having the system explicitly support transformation rules has several important benefits:

1) All assumptions implicit in a transformation can be automatically checked each time the transformation is applied. If the tool does not have enough information to verify some assumptions, the user can be intelligently prompted using easily understood questions, for instance: "Are the values of attribute X always identical to attribute Y?" or "Is every instance of class X associated with an instance of class Y?".

2) Where more than one transformation can be applied, and there is not enough information to determine which is appropriate, the tool can ask the user, and offer meaningful suggestions. For example, consider the schemas of Figure 7. Suppose $O1$ is the set of graduate students and $O2$ the set of university employees and that both are identified by their university id numbers. In Schema I, graduate students have been represented as a subclass of employees (denoted by the selection edge σ), while in Schema II there is a functional relationship (which the user may have labeled "is-a") from students to employees. Rather than asking the designer which schema is correct, a tool could ask the designer if any employee in Schema II can be related to more than one graduate student (that is, whether the "is-a" edge is injective). If not, the next question should be whether a student can be related to an employee with a different id number. If the answer to both questions is negative, then the choice of Schema I as the integrated schema is appropriate. Otherwise, the tool must consider the possibility that Schema I is incorrectly specified or that these two relationships are not identical.

3) As we illustrated in earlier sections, using transformation rules at the schema level can lead to automatic query transformation as well.

4) The sequence of transformation steps in an actual integration may be automatically recorded and serve as a history of the integration. Such a record is valuable both during the course of integration (e.g., to check assumptions made thus far) and subsequently, as meta-information about the integrated schema.

7.3 The Role of Information Capacity

In order to realize the full potential of a system that supports schema transformations, it is important to thoroughly understand the assumptions associated with a rule and the mappings between schema elements. We have argued that a good way to obtain such an understanding is to analyze these rules from the standpoint of information capacity. A potential further benefit, of course, is that in addition to achieving a sound integration at the schema level, we also obtain a sound query-level mapping.

We have studied several proposed transformations but we have not come close to exhausting the possible transformations that a designer may formulate in the context of a specific integration task, based on a semantic understanding of the data. However, if a designer proposes a new transformation and provides the corresponding instance mapping, a tool may still make use of this information in combination with the automated trans-

formations. Such ad hoc transformations may therefore be integrated into our methodology. Given a fully specified ad hoc transformation, translation of queries and instances may still be done automatically. (An interesting question is whether a tool can be developed to examine arbitrary schema transformations from the point of view of information capacity, and automatically identify missing assumptions and mappings.)

8 Conclusions and Future Work

We have examined schema integration and translation tasks and proposed a definition of correctness that accounts for the different goals of related tasks. By analyzing the correctness of proposed schema transformations, we have shown how potentially severe anomalies can be avoided. Additionally, we have demonstrated how the notion of correctness provides an opportunity for increased intelligence in an interactive schema transformation tool.

References

[AABM82] P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and Equivalence between Relational Database Schemata. *Theoretical Computer Science*, 19:267–285, 1982.

[ADD+91] R. Ahmed, P. DeSmedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, and M. C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, pages 19–26, December 1991.

[AP82] P. Atzeni and D. S. Parker. Assumptions in Relational Database Theory. In *Proc. of Conf. on PODS*, pages 1–9, Los Angeles, CA, March 1982.

[AP86] P. Atzeni and D. S. Parker. Formal Properties of Net-Based Knowledge Representation Schemes. In *Proc. of Data Eng. Conf.*, pages 700–706, 1986.

[BC86] J. Biskup and B. Convent. A Formal View Integration Method. In *Proc. of SIGMOD Conf.*, pages 398–407, Washington, D.C., May 1986.

[BDK92] P. Buneman, S. Davidson, and A. Kosky. Theoretical Aspects of Schema Merging. In *Proc. of EDBT Conf.*, pages 152–167, 1992.

[BL84] C. Batini and M. Lenzerini. A Methodology for Data Schema Integration in the Entity Relationship Model. *IEEE Trans. on Software Eng.*, SE-10(6):650–664, November 1984.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[Bor78] S. A. Borkin. Data Model Equivalence. In *Proc. of Conf. on VLDB*, pages 526–534, 1978.

[DH84] U. Dayal and H. Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Trans. on Software Eng.*, SE-10(6):628–644, November 1984.

[EHW85] R. Elmasri, A. Hevner, and J. Weldreyer. The Category Concept: An Extension to the Entity-Relationship Model. *Data and Knowledge Eng.*, 1(1):75–116, June 1985.

[Eic91] C. F. Eick. A Methodology for the Design and Transformation of Conceptual Schemas. In *Proc. of Conf. on VLDB*, pages 25–34, Barcelona, Spain, 1991.

[Hul86] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal of Computing*, 15(3):856–886, August 1986.

[HY84] R. Hull and C. K. Yap. The Format Model: A Theory of Database Organization. *J. ACM*, 31(3):518–537, 1984.

[Kal90] L. A. Kalinichenko. Methods and Tools for Equivalent Data Model Mapping Construction. In *Proc. of EDBT Conf.*, pages 92–119, 1990.

[LNE89] J. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Trans. on Software Eng.*, 15(4):449–463, April 1989.

[MB81] A. Motro and P. Buneman. Constructing Superviews. In *Proc. of SIGMOD Conf.*, pages 56–64, 1981.

[ME84] M. V. Mannino and W. Effelsberg. Matching Techniques in Global Schema Design. In *Proc. of Data Eng. Conf.*, pages 418–425, 1984.

[MIR] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice. *Submitted for Publication.*

[Mot87] A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Trans. on Software Eng.*, 13(7):785–798, July 1987.

[MS92] V. M. Markowitz and A. Shoshani. Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach. *ACM Trans. on DB Systems*, 17(3):423–464, September 1992.

[NEL86] S. B. Navathe, R. Elmasri, and J. Larson. Integrating User Views in Database Design. *IEEE Computer*, 19(1):50–62, January 1986.

[Ris82] J. Rissanen. On Equivalences of Database Schemes. In *Proc. of Conf. on PODS*, pages 23–26, 1982.

[RR87] A. Rosenthal and D. Reiner. Theoretically Sound Transformations for Practical Database Design. In Salvatore T. March, editor, *Proc. of Entity-Relationship Conf.*, pages 115–131, NYC, NY, 1987. Elsevier Science Pub.

[RR89] A. Rosenthal and D. Reiner. Database Design Tools: Combining Theory, Guesswork, and User Interaction. In *Proc. of Entity-Relationship Conf.*, pages 391–405, Toronto, Canada, 1989.

[SZ91] P. Shoval and S. Zohn. Binary-Relationship Integration Methodology. *Data and Knowledge Eng.*, 6:225–250, 1991.

[Tro93] O. De Troyer. *On Data Schema Transformation.* PhD thesis, Katholieke Universiteit Brabant, Netherlands, 1993.

[WE79] G. Wiederhold and R. Elmasri. The Structural Model for Database Design. In P. P. Chen, editor, *Entity-Relationship Conf.*, pages 237–257, Los Angeles, CA, December 1979. North-Holland Pub. Co.