Pergamon

Information Systems Vol. 23, No. 5, pp. 307–334, 1998

Printed in Great Britain
0306-4379/98 $17.00 + 0.00

A FORMALISATION OF SEMANTIC SCHEMA INTEGRATION†

PETER MCBRIEN AND ALEXANDRA POULOVASSILIS

Department of Computer Science, King's College London, Strand, London WC2R 2LS, U.K.

(Received 6 January 1996; in final revised form 8 April 1998)

Abstract — Several methodologies for the semantic integration of databases have been proposed in the literature. These often use a variant of the Entity-Relationship (ER) model as the common data model. To aid the schema conforming, merging and restructuring phases of the semantic integration process, various transformations have been defined that map between ER representations which are in some sense equivalent. Our work aims to formalise the notion of schema equivalence and to provide a formal underpinning for the schema integration process. We show how **transformational**, **mapping** and **behavioural schema equivalence** are all variants of a more general definition of schema equivalence. We propose a semantically sound set of primitive transformations and show how they can be used to express the transformations commonly used during the schema integration process and to define new transformations. We differentiate between transformations which apply to any instance of a schema and those which require knowledge-based reasoning since they apply only for certain instances; this distinction could serve to enhance the performance of transformation tools since it identifies which transformations must be verified by inspection of the schema extension; it also serves to identify when intelligent reasoning is required during the schema integration process. Copyright ©1998 Elsevier Science Ltd

Key words: Schema Equivalence, Schema Transformation, Schema Integration

1. INTRODUCTION

The now widespread use of computers for data-processing in large and distributed organisations means that it is often the case that one organisation will hold data in a variety of database systems, ranging from those based on flat files, hierarchical or relational models through to those based on object-oriented models. When data is to be shared or exchanged between such databases, it is necessary to build a single integrated schema — a *federal schema* [19] — expressed using a *common data model* (CDM). The task of *database schema integration* involves unifying several export schemas. Due to variations in the way different databases will have been modelled during design, it is usually the case that conflicts exist, in that the same concept in the universe of discourse is represented in different ways, or different concepts are represented in the same way. Schema integration must remove such conflicts by performing *transformations* on the export schemas to produce equivalent schemas which may participate in a database federation.

In this paper we propose a new formal definition for *schema equivalence* which subsumes several previous notions of this concept. We define a semantically sound set of primitive transformations and show how they can express the transformations commonly used in the schema integration process, as well as new transformations. Along the way we clarify what kind of schema equivalence the transformations commonly found in the literature obey. A detailed comparison of our approach with related work in given in Section 7. We continue this introductory section by defining the ER model that we use as the CDM (Section 1.1) and by reviewing the schema integration process and previous notions of schema equivalence (Section 1.2). We give an outline of the remainder of the paper Section 1.3.

1.1. A Binary Entity-Relationship CDM

Our formal framework for schema integration assumes as the CDM a binary ER model with subtypes. This model supports as primitive constructs entity types with attributes, subtype relationships between entity types and binary relationships (without attributes) between entity types.

†Recommended by Maurizio Lenzerini

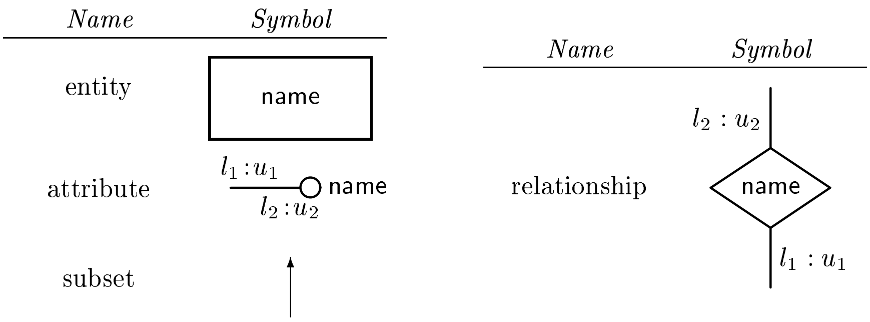


Table 1: Constructs of the CDM

This choice of CDM may appear dated in the light of the current trend towards OO modelling, but our aim is to draw together and formalise several notions of equivalence that have been described in the past, and these have often be based on ER modelling [3, 4, 7]. The popularity and prevalence of the ER model in the commercial sector also increases the practical appeal of our work.

By supporting subtype relationships we achieve a *structurally object-oriented* model *i.e.* one which is able to represent the data types, but not the methods, that are found in OO languages. Thus the model has sufficient modelling expressiveness for the task of schema integration of OO data models. Omitting generalisation hierarchies, aggregations and n-ary relationships for n > 2 as CDM primitives avoids some alternate representations of the universe of discourse. There is no loss of modelling expressiveness and Section 6 discusses how we may use our ER model to handle these constructs.

Table 1 gives the graphical syntax we use for our CDM, which is a subset of the notation used in [3, 4, 7]. The notation of [3] is used for specifying cardinality constraints. Thus, a relationship between two entity types e_1, e_2 has a pair of cardinality constraints associated with it, $l_1 : u_1$ and $l_2 : u_2$, where $l_1 : u_1$ (resp. $l_2 : u_2$) indicates the lower and upper cardinalities of the involvement in the relationship by each instance of e_1 (resp. e_2). If a cardinality constraint is not stated then it is assumed to be $1 : 1$. Similarly, each association between an entity type and an attribute has a pair of cardinality constraints associated with it.

1.2. Background on Schema Transformation, Integration and Equivalence

Database *schema integration* consists of the following main tasks [4]:

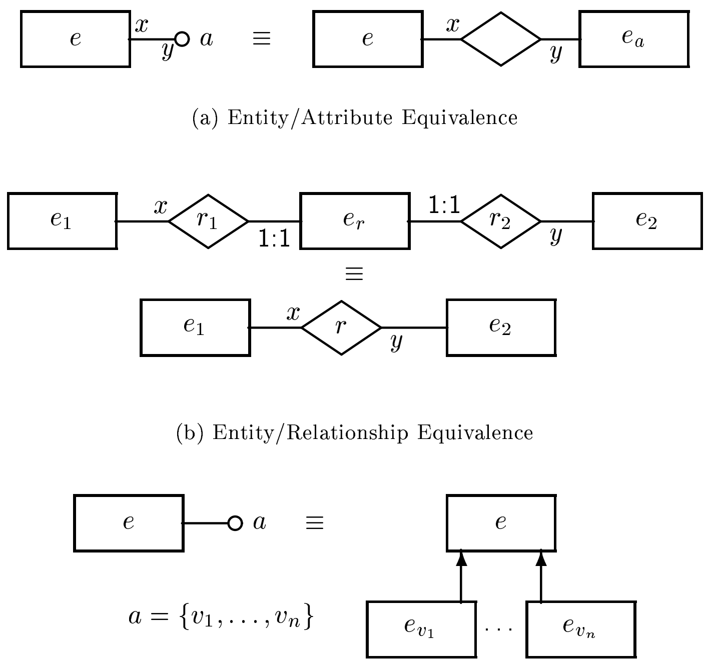
(i) *pre-integration* where the export schema is created and the elements to be integrated are selected to form the component schemas;

(ii) *schema conforming* during which *conflict detection* and *conflict resolution* are performed, resulting in new versions of the component schemas which represent the same concepts in the same manner;

(iii) *schema merging* where related concepts are identified and a single federated schema is produced; and

(iv) *schema restructuring* during which the quality of the federated schema is improved; in particular, redundant information is removed so that the resulting model is the 'minimum' union of the component schemas.

Performing all but the first of these tasks involves a process of *schema transformation* during which schemas are manipulated in such a manner that the resulting schemas can be regarded as in some sense *equivalent* to the original ones. Figures 1–3 illustrate some transformations commonly used for each task.



(a) Entity/Attribute Equivalence

(b) Entity/Relationship Equivalence

(c) Attribute/Subtype Equivalence

Fig. 1: Schema Conforming Transformations

Three categories of schema equivalence are frequently identified in the literature [4, 7]:

(i) *transformational equivalence* [3] exists between two schemas S and S' if there is a sequence of primitive reversible transformations which may be applied to S to produce S'. Each of the primitive transformations is by definition correct and should preserve the instances of the schema.

For example, an attribute is regarded as equivalent to an attribute-less entity that participates in precisely one relationship (see Figure 1(a)). Also, a many-many relationship is regarded as equivalent to two one-many ones (see Figure 1(b)).

(ii) *mapping equivalence* [17, 12, 15] exists between two schemas S and S' if for any pair of instances of S and S' there is a one-to-one correspondence between their elements.

For example, the schemas of Figure 1(a) are mapping equivalent but the schemas of Figure 1(b) are not. However, mapping equivalence is not subsumed by transformation equivalence since with mapping equivalence knowledge about the *instances* of a given schema may be utilised. For example, the transformations illustrated in Figure 3 are mapping equivalences in which knowledge about the instances of the left-hand schema is utilised;

(iii) *behavioural equivalence* [2, 6] exists between two schemas S and S' if for any query Q over an instance I of schema S there exists a transformation of S to S', I to an instance I' of S' and Q to a query Q' over I' such that Q and Q' have the same result, and vice versa for any query Q' over an instance I' of schema S'.

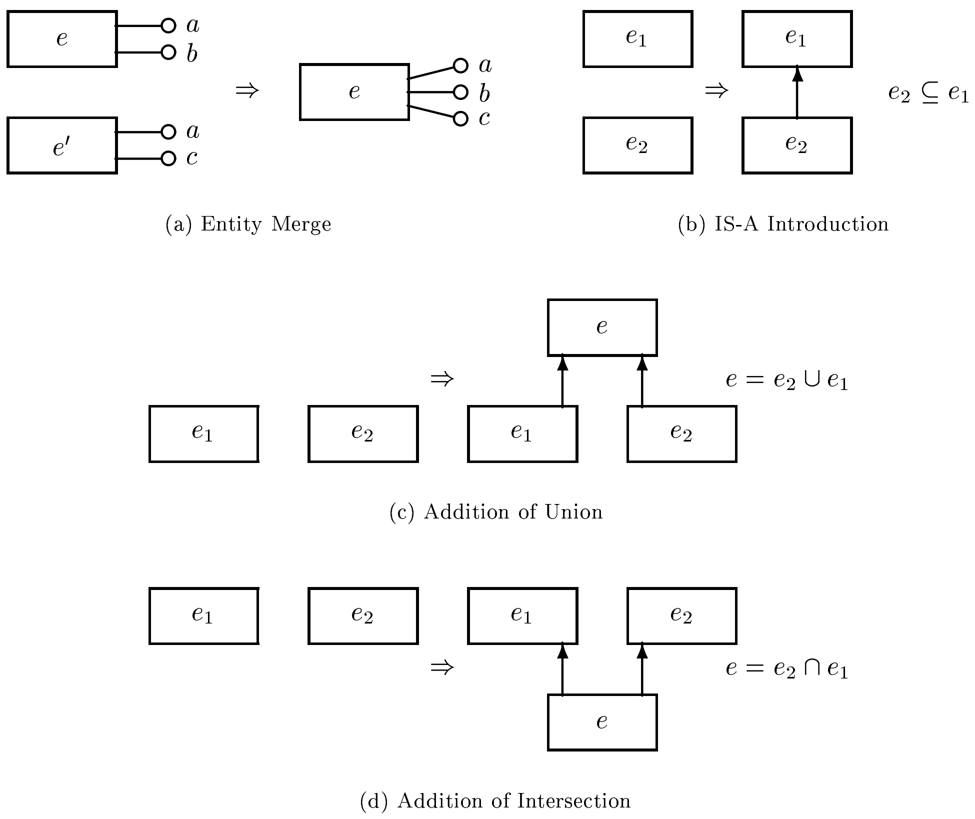
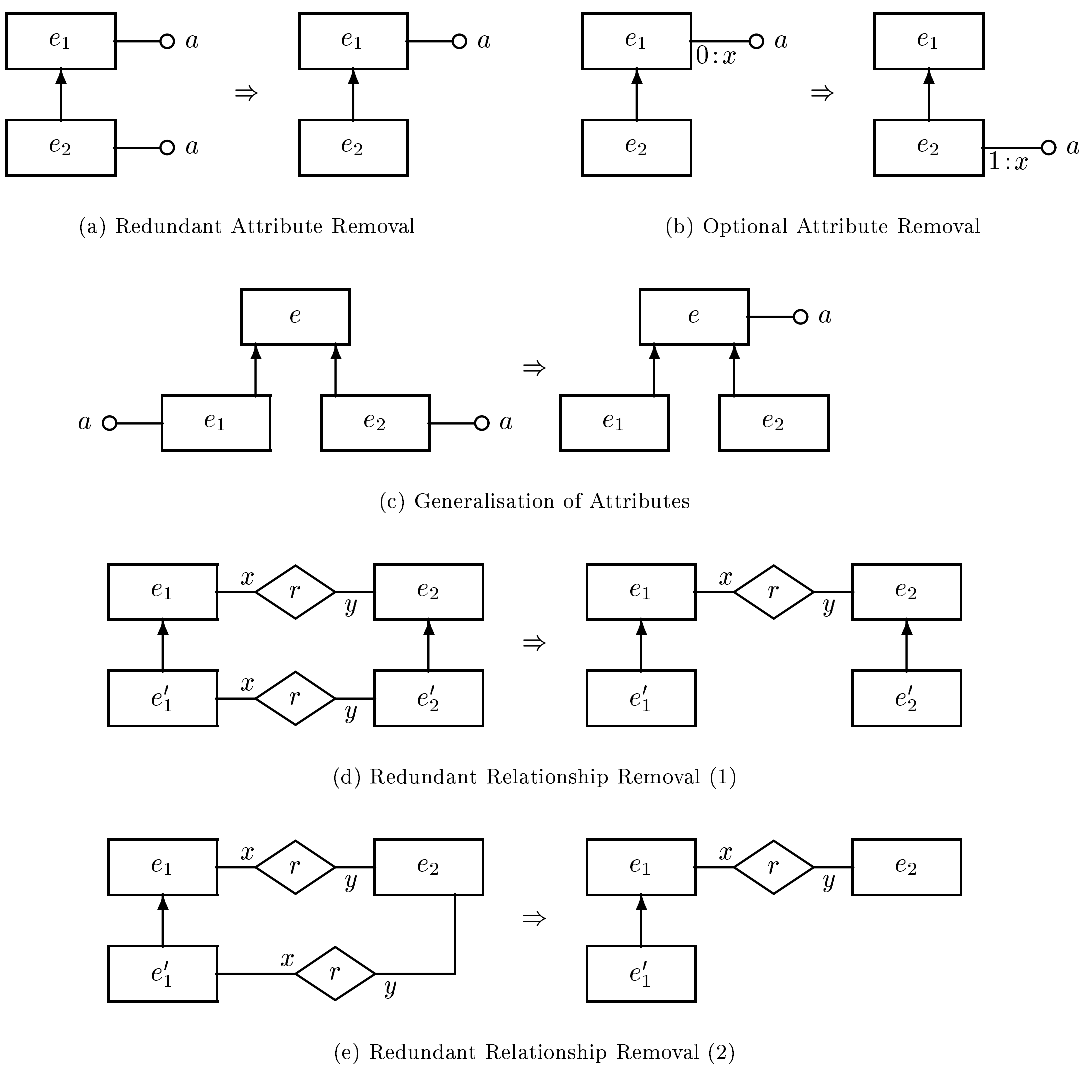


Fig. 2: Schema Merging Transformations

> This notion of equivalence is the most general of the three, since it allows transformations which are dependent on knowledge relating to the possible instances of schemas, but which do not necessarily produce a one-to-one mapping between pairs of instances. For example, we may divide an entity type into sub-types using an is-a hierarchy by utilising knowledge about the domain of an attribute associated with the entity type (see Figure 1(c)).

Table 2 classifies the transformations of Figures 1–3 according to which notion of equivalence they obey, where TE, ME and BE respectively denote transformational, mapping and behavioural equivalence. We illustrate how this classification is obtained by considering the entity/relationship equivalence of Figure 1(b). Firstly, it is possible to derive a relationship r from an entity e_r by setting r to be the join of r_1 and r_2 (which is lossless because of the cardinalities shown); conversely, it is possible to derive an entity e_r and relationships r_1 and r_2 from a relationship r by setting all of e_r, r_1 and r_2 to be r; thus the equivalence is TE. Secondly, the equivalence is not ME since three elements of the upper schema (e_r, r_1 and r_2) correspond to one element of the lower (r). Finally, the equivalence is BE since the two transformations given for TE can be used to transform a query over one schema to a query giving the same result over the other schema.

Each of the above notions of schema equivalence has lead to competing methodologies for the task of schema integration. What we aim to achieve in this paper is a formalisation which may be used as a foundation for any of these notions of equivalence. Moreover our approach clearly differentiates the type of equivalence that each transformation preserves since each transformation is classified into precisely one category, as opposed to possibly several categories as in Table 2.



(a) Redundant Attribute Removal

(b) Optional Attribute Removal

(c) Generalisation of Attributes

(d) Redundant Relationship Removal (1)

(e) Redundant Relationship Removal (2)

Fig. 3: Schema Restructuring Transformations

Integration Stage	Transformation Description	TE	ME	BE
Schema conform (Figure 1)	Entity/attribute	√	√	√
	Entity/relationship	√		√
	Attribute/is-a			√
Schema merge (Figure 2)	Entity merge		√	√
	is-a introduction		√	√
	Addition of union	√	√	√
	Addition of intersection	√	√	√
Schema restructuring (Figure 3)	Redundant attribute removal		√	√
	Optional attribute removal		√	√
	Generalisation of attributes		√	√
	Relationship removal (1)		√	√
	Relationship removal (2)		√	√

Table 2: Classification of Common Schema Transformations

There have been many previous definitions of schema equivalence, which we compare in detail with our approach in Section 7. The difference between our approach and that exemplified by work such as [15, 5, 21] is that we can make our schema equivalences conditional on the schema only having certain extensions (or 'real world states') *without* the necessity of adding extra constraint rules to the data model, as is found in [5, 21] for example. In contrast, our approach is more general since constraints are incorporated as boolean-valued expressions as part of the equivalence rules, thus allowing arbitrary constraints to be expressed.

1.3. Outline of the Paper

In Section 2 we introduce the basic concepts of our approach, namely schemas, instances, models, and schema equivalence, and we illustrate them with suitable examples. In Section 3 we give a set of instance-preserving primitive transformations and explore their properties. We then discuss composite transformations, which are sequences of primitive transformations, and knowledge-based transformations, which allow necessary conditions on instances to be expressed. In Section 4 we discuss how our formalism can be used to underpin the schema integration process and in Section 5 we show how the formalism can be used in practice to define new schema transformations. In Section 6 we discuss how the formalism can handle richer modelling constructs such as n-ary relationships, complex attributes and generalisation hierarchies. In Section 7 we compare our approach with previous work on the formalisation of schema equivalence. We also show how our definition of schema equivalence generalises the notions of transformational, mapping and behavioural equivalence, and we give an alternative classification of the transformations of Table 2 in which each transformation obeys precisely one type of equivalence. Section 8 gives our concluding remarks.

2. THE FORMALISM

2.1. ER Schemas, Instances and Models

Before defining these notions we require some auxiliary definitions. Firstly, we assume the availability of two disjoint sets, $Vals$ (values) and $Names$ (the names of entity-types, attributes and relationships). We denote by $Seq(Vals)$ the set of sequences of values of arbitrary (but finite) length. Since we are interested in merging multiple schemas, each $n \in Names$ is assumed to have two parts, a **schema identifier**, which we denote by $Schema(n)$, and an **item identifier**, which we denote by $Item(n)$. The schema identifier is unique for each schema and is common to all items of that schema. The distinguished constant $Null$ is a valid item identifier. We also require a set of cardinality constraints $Cards$, a cardinality constraint being a pair $lower : upper$, where $lower$ is a natural number and *upper* is either a natural number or N (denoting no upper limit). Finally, for any function $f : D \to E$, we denote by $Range(f)$ the subset of E that is the range of f.

Definition 1 An *ER schema*, S, is a quadruple $\langle Ents, Incs, Atts, Assocs \rangle$ where:

- $Ents \subseteq Names$ is the set of entity-type names.
- $Incs \subseteq (Ents \times Ents)$, each pair $\langle e_1, e_2 \rangle \in Incs$ representing that e_1 is a subtype of e_2. We assume that $Incs$ is acyclic.
- $Atts \subseteq Names$ is the set of attribute names.
- $Assocs \subseteq (Names \times Names \times Names \times Cards \times Cards)$ is the set of *associations*, where:
  (i) For each binary relationship between two entity types $e_1, e_2 \in Ents$, there is a tuple in $Assocs$ of the form:
  $$\langle rel\_name, e_1, e_2, c_1, c_2 \rangle$$
  c_1 indicates the lower and upper cardinalities of instances of e_2 for each instance of e_1 while c_2 indicates the lower and upper cardinalities of instances of e_1 for each instance of e_2. rel_name may be $Null$ if there is only one relationship between e_1 and e_2.
  We call the triple $\langle rel\_name, e_1, e_2 \rangle$ the *scheme* of the binary relationship and note that it uniquely identifies the relationship.
  (ii) For each attribute a associated with an entity type e there is a tuple in $Assocs$ of the form:
  $$\langle Null, e, a, c_1, c_2 \rangle$$
  c_1 indicates the lower and upper cardinalities of a for each instance of e, and c_2 indicates the lower and upper cardinalities of instances of e for each value of a.
  We call the triple $\langle Null, e, a \rangle$ the *scheme* of the association. Since an entity type can be related to an attribute in only one way, this scheme uniquely identifies the association within the set $Assocs$.

We define the notion of an *instance* of a schema in Definition 2 below. From this definition we see that this is not an absolute notion but depends on the expressiveness of the language, L, that maps between the conceptual schema and the database extension. In particular, an instance I is a set of sets. The extent of each item in the schema can be derived by means of an expression in L over the sets of I. In order to support updates to I, this mapping must be reversible, in the sense that each set of I can be derived by means of some expression in L over the extents of the schema items †. The instance I must also respect the subtype relationships in the schema, and the domain and cardinality constraints:

Definition 2 Given an ER schema $S = \langle Ents, Incs, Atts, Assocs \rangle$, let

$$Schemes = \{\langle n_0, n_1, n_2 \rangle \mid \langle n_0, n_1, n_2, c_1, c_2 \rangle \in Assocs\}$$

An *instance of* S is a set $I \subseteq P(Seq(Vals))$ such that there exists a function

$$Ext_{S,I} : Ents \cup Atts \cup Schemes \to P(Seq(Vals))$$

where:

(i) each set in $Range(Ext_{S,I})$ is derivable by means of an expression in L over the sets of I;

(ii) conversely, each set in I is derivable by means of an expression in L over the sets of $Range(Ext_{S,I})$;

† In practical database systems these expressions will be *C-generic* [1]. Informally, this means that apart from the set of constants C appearing in the expression, only the structure of the input sets is relevant, not their content.

(iii) for any $\langle e_1, e_2\rangle \in Incs$, $Ext_{S,I}(e_1) \subseteq Ext_{S,I}(e_2)$;

(iv) for every $\langle n_0, n_1, n_2\rangle \in Schemes$, $Ext_{S,I}(\langle n_0, n_1, n_2\rangle)$ satisfies the appropriate domain and cardinality constraints *i.e.* the following two conditions hold:

(a) each sequence $s \in Ext_{S,I}(\langle n_0, n_1, n_2\rangle)$ contains two, possibly overlapping, subsequences s_1, s_2 where $s_1 \in Ext_{S,I}(n_1)$ and $s_2 \in Ext_{S,I}(n_2)$; we denote by $n_1(s)$ and $n_2(s)$ these subsequences of s;

(b) if the cardinality associated with $\langle n_0, n_1, n_2\rangle$ is $\langle l_1 : u_1, l_2 : u_2\rangle$, then

$$\forall s_1 \in Ext_{S,I}(n_1). l_1 \leq | \{n_2(s) \mid s \in Ext_{S,I}(\langle n_0, n_1, n_2\rangle) \ \wedge \ n_1(s) = s_1\} | \leq u_1$$

and

$$\forall s_2 \in Ext_{S,I}(n_2). l_2 \leq | \{n_1(s) \mid s \in Ext_{S,I}(\langle n_0, n_1, n_2\rangle) \ \wedge \ n_2(s) = s_2\} | \leq u_2$$

We call a function $Ext_{S,I}$ an *extension mapping* from S to I.

Definition 3 We define a *model* to be a triple $\langle S, I, Ext_{S,I}\rangle$ where S is a schema, I is an instance of S and $Ext_{S,I}$ is an extension mapping from S to I. We denote by $Models$ the set of models. For any schema, S, *a model of* S is a model which has S as its first component.

2.2. Unconditional Equivalence of ER Schemas

Definition 4 We denote by $Inst(S)$ the set of instances of a schema S. A schema S *subsumes* a schema S' if $Inst(S') \subseteq Inst(S)$. Two schemas S and S' are *unconditionally equivalent* (*u-equivalent*) if $Inst(S') = Inst(S)$.

We observe that, as with instances, u-equivalence is not an absolute notion but depends on the expressiveness of the mapping language, L, between the conceptual schema and the database extension. We also observe that the above definition implies that if S and S' are u-equivalent then each extension of S can be derived from an extension of S', and vice versa. To see why this is so, let $S = \langle Ents, Atts, Incs, Assocs\rangle$ and $S' = \langle Ents', Atts', Incs', Assocs'\rangle$, and consider two models both with the same instance component, $\langle S, I, Ext_{S,I}\rangle$ and $\langle S', I, Ext_{S',I}\rangle$. By the definition of an instance, for each item $o \in Ents \cup Atts \cup Schemes$, $Ext_{S,I}(o) = expr_o$ for some expression $expr_o$ in L over I. Also, for each set $i \in I$, $i = expr_i$ for some expression $expr_i$ in L over $Range(Ext_{S',I})$. Thus, each set in $Range(Ext_{S,I})$ can be derived from $Range(Ext_{S',I})$ by means of an expression in L. By a similar argument, each set in $Range(Ext_{S',I})$ can be derived from $Range(Ext_{S,I})$.

We illustrate this in the following example, which formalises the schema transformations of Figure 1 (a) and (b). In this, and all other examples in the paper, we assume that the mapping language L is the relational calculus.

Example 1 (a) Let $S = \langle\{e\}, \{\}, \{a\}, \{\langle Null, e, a, c_1, c_2\rangle\}\rangle$ and $S' = \langle\{e, e_a\}, \{\}, \{\}, \{\langle Null, e, e_a, c_1, c_2\rangle\}\rangle$.

Then for any instance I of S and mapping $Ext_{S,I}$ define $Ext_{S',I}$ as follows:

$$\begin{array}{rcl} Ext_{S',I}(e) & = & Ext_{S,I}(e) \\ Ext_{S',I}(e_a) & = & Ext_{S,I}(a) \\ Ext_{S',I}(\langle Null, e, e_a\rangle) & = & Ext_{S,I}(\langle Null, e, a\rangle) \end{array}$$

From this it is easy to verify that I is an instance of S'.

Conversely, for any instance I of S' we can similarly derive $Ext_{S,I}$ from $Ext_{S',I}$. Thus S and S' are u-equivalent.

(b) Let $S = \langle\{e_1, e_r, e_2\}, \{\}, \{\}, \{\langle r_1, e_1, e_r, c_1, 1{:}1\rangle, \langle r_2, e_r, e_2, 1{:}1, c_2\rangle\}\rangle$ and $S' = \langle\{e_1, e_2\}, \{\}, \{\}, \{\langle r, e_1, e_2, c_1, c_2\rangle\}\rangle$.

Then for any instance I of S and mapping $Ext_{S,I}$ define $Ext_{S',I}$ as follows:

$$\begin{array}{rcl} Ext_{S',I}(e_1) & = & Ext_{S,I}(e_1) \\ Ext_{S',I}(e_2) & = & Ext_{S,I}(e_2) \\ Ext_{S',I}(\langle r, e_1, e_2\rangle) & = & \{\langle e_1(s_1), e_r(s_1), e_2(s_2)\rangle \mid s_1 \in Ext_{S,I}(\langle r_1, e_1, e_r\rangle) \ \wedge \\ & & \qquad s_2 \in Ext_{S,I}(\langle r_2, e_r, e_2\rangle) \wedge e_r(s_1) = e_r(s_2)\} \end{array}$$

From this it is easy to verify that I is an instance of S'.

Conversely, for any instance I of S' and mapping $Ext_{S',I}$ define $Ext_{S,I}$ as follows:

$$\begin{array}{rcl} Ext_{S,I}(e_1) & = & Ext_{S',I}(e_1) \\ Ext_{S,I}(e_2) & = & Ext_{S',I}(e_2) \\ Ext_{S,I}(e_r) & = & Ext_{S',I}(\langle r, e_1, e_2\rangle) \\ Ext_{S,I}(\langle r_1, e_1, e_r\rangle) & = & Ext_{S',I}(\langle r, e_1, e_2\rangle) \\ Ext_{S,I}(\langle r_2, e_r, e_2\rangle) & = & Ext_{S',I}(\langle r, e_1, e_2\rangle) \end{array}$$

From this it is easy to verify I is an instance of S. Thus S and S' are u-equivalent. □

2.3. Conditional Equivalence of ER Schemas

Definition 5 Given a condition, f, we denote by $Inst(S, f)$ the set of instances of a schema S that satisfy f. Two schemas S and S' are *conditionally equivalent (c-equivalent) w.r.t* f if $Inst(S, f) = Inst(S', f)$.

Thus c-equivalence generalises u-equivalence by incorporating a condition on the instances of one or both schemas. We illustrate c-equivalence by considering the schema transformation of Figure 1 (c):

Example 2 Let $S = \langle\{e\}, \{a\}, \{\}, \{\langle Null, e, a, m_1{:}n_1, m_2{:}n_2\rangle\}\rangle$ and $S' = \langle\{e, e_1, \ldots, e_n\}, \{\}, \{\langle e_1, e\rangle, \ldots, \langle e_n, e\rangle\}, \{\}\rangle$. Let the condition f be

$$Ext_{S,I}(a) = \{v_1, \ldots, v_n\}$$

Then for any instance $I \in Inst(S, f)$ and mapping $Ext_{S,I}$, we can define $Ext_{S',I}$ as follows:

$$\begin{array}{rcl} Ext_{S',I}(e) & = & Ext_{S,I}(e) \\ Ext_{S',I}(e_i) & = & \{e(s) \mid s \in Ext_{S,I}(\langle Null, e, a\rangle) \wedge a(s) = v_i\} \end{array}$$

from which we see that $I \in Inst(S')$. Since f is vacuously true for all instances of S', we have $I \in Inst(S', f)$. Conversely, let the condition f' be

$$(\forall x \in Ext_{S',I}(e).m_1 \leq |\ \{e_i \mid x \in Ext_{S',I}(e_i)\}\ | \leq n_1) \ \wedge$$
$$(\forall e_i \in \{e_1, \ldots, e_n\}.m_2 \leq |\ Ext_{S',I}(e_i)\ | \leq n_2)$$

Then for any instance $I \in Inst(S', f')$ and mapping $Ext_{S',I}$, we can define $Ext_{S,I}$ as follows:

$$\begin{array}{rcl} Ext_{S,I}(e) & = & Ext_{S',I}(e) \\ Ext_{S,I}(a) & = & \{v_1, \ldots, v_n\} \\ Ext_{S,I}(\langle Null, e, a\rangle) & = & \cup_{i=1\ldots n}\{\langle x, v_i\rangle \mid x \in Ext_{S',I}(e_i)\} \end{array}$$

from which we see that $I \in Inst(S)$. Since f' is vacuously true for all instances of S, we have $I \in Inst(S, f')$. Since f' is true of S' whenever f is true of S, and f is true of S whenever f' is true of S', we have that S and S' are c-equivalent w.r.t. $f \wedge f'$. □

3. TRANSFORMATION OF ER MODELS

We now use the above definitions of u- and c-equivalence as the semantic foundation for a set of primitive transformations on ER models. Primitive transformations may always be applicable to an ER schema irrespective of the instance, or may only be applicable if the instance satisfies certain conditions. We call the former *schema-dependent* (s-d) transformations and the latter *instance-dependent* (i-d) ones. We show that if a schema S can be transformed to a schema S' by a sequence of s-d primitive transformations, and vice versa, then S and S' are u-equivalent. We show an analogous result for i-d primitive transformations and c-equivalence. We then enhance our primitive transformations by allowing them to take as an extra parameter a condition on the model which must be satisfied in order for the transformation to be applicable. We call these transformations *knowledge-based* (k-b) ones. We illustrate by showing how some of the equivalences of Figures 1–3 can be expressed using s-d, i-d or k-b transformations.

3.1. Primitive Transformations

Each primitive transformation is a function of type $ArgType \rightarrow Models \rightarrow Models$ for some type $ArgType$ *i.e.* it takes a model and an argument of type $ArgType$ and returns a new model. The instance component of the input model is left unchanged by the transformation, and only the schema and extension mapping components may change.

Each primitive transformation has a number of provisos associated with it. If these are not met then the transformation is assumed to return an 'undefined' value, which we denote by ϕ. The result of applying a primitive transformation to ϕ is assumed to be ϕ.

Definition 6 below states the name and the $ArgType$ of each primitive transformation and informally describes its effect. The formal definition of each transformation, including the full set of provisos associated with it, are given in Definition 11 in Appendix A. In Definition 6, $Queries$ denotes the set of *queries* expressible in L where, given a schema $S = \langle Ents, Atts, Incs, Assocs\rangle$ and a model $\langle S, I, Ext_{S,I}\rangle$, a *query* Q *over* $\langle S, I, Ext_{S,I}\rangle$ is an expression in L whose variables, $v_1, \ldots, v_n$, are members of $Ents \cup Atts \cup Schemes$. The *value* of Q is given by the expression $Q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$. Queries are used by transformations that add a new item to a schema, in that the query is applied to the items of the input schema and returns the extent of the new item. Thus, as with our notions of instances and schema equivalence, transformations which add new schema items are language-dependent.

Definition 6 The primitive transformations are as follows:

1. $rename_X(Names \times Names)$ renames an entity, attribute or relationship.

   The subscript X may be E, A or R, denoting entity, attribute or relationship, respectively. $rename_X$ takes a pair $\langle fromName, toName\rangle$ and a model and returns a new model in which all occurrences of $fromName$ in the schema are replaced by $toName$.

2. $expand(Names \times Names \times Names \times Cards \times Cards)$ expands a constraint pair.

   This takes a tuple of the form $\langle n_0, n_1, n_2, c_1, c_2\rangle$ and a model, and replaces the cardinality constraint on the scheme $\langle n_0, n_1, n_2\rangle$ by the new, relaxed, constraint $c_1 : c_2$.

3. $contract(Names \times Names \times Names \times Cards \times Cards)$ contracts a constraint pair.

   This similarly takes a tuple of the form $\langle n_0, n_1, n_2, c_1, c_2\rangle$ and a model, and replaces the cardinality constraint on the scheme $\langle n_0, n_1, n_2\rangle$ by the new, stricter, constraint $c_1 : c_2$.

4. $add_E(Names \times Queries)$ adds a new entity.

   This takes a pair of the form $\langle e, q\rangle$ and a model, and adds e to the entity set of the schema. The extent of the new entity e is given by the value of the query q.

5. $del_E(Names)$ deletes an entity.

This takes an entity e and a model, and removes e from the entity set of the schema, provided that e has no attributes and participates in no relationships and that property (ii) of Definition 2 is not violated by setting the extent of e to be undefined†.

6. $add_R(Names \times Names \times Names \times Cards \times Cards \times Queries)$ adds a new relationship.

   This takes a tuple $\langle r, e_1, e_2, c_1, c_2, q\rangle$ and a model, and adds the relationship $\langle r, e_1, e_2, c_1, c_2\rangle$ to the set of associations of the schema. The extent of the new relationship is given by the value of the query q.

7. $del_R(Names \times Names \times Names)$ removes a relationship.

   This takes a tuple of the form $\langle r, e_1, e_2\rangle$ and a model, and removes the relationship with scheme $\langle r, e_1, e_2\rangle$ from the set of associations of the schema, provided that property (ii) of Definition 2 is not violated by setting the extent of this scheme to be undefined.

8. $add_A(Names \times Names \times Cards \times Cards \times Queries \times Queries)$ adds an attribute to an entity type, where the attribute may either already exist in $Assocs$ or may be a new attribute.

   This takes a tuple of the form $\langle e, a, c_1, c_2, q_{att}, q_{assoc}\rangle$ and a model, and adds the association $\langle Null, e, a, c_1, c_2\rangle$ to the set of associations of the schema. The extent of the attribute is given by the value of the query q_{att} and the extent of the association is given by the value of the query q_{assoc}.

9. $del_A(Names \times Names)$ removes an attribute from an entity type.

   This takes a pair of the form $\langle e, a\rangle$ and a model, and removes the association with scheme $\langle Null, e, a\rangle$ from the set of associations of the schema, provided that property (ii) of Definition 2 is not violated.

10. $add_I(Names \times Names)$ adds a subtype relationship.

    This takes a pair of entities $\langle e_1, e_2\rangle$ and a model, and adds this subtype relationship to the schema, provided that the extent of e_1 is indeed contained in the extent of e_2.

11. $del_I(Names \times Names)$ removes a subtype relationship.

    This takes a pair of entities $\langle e_1, e_2\rangle$ and a model and removes this subtype relationship from the schema.

We make the following observations about the primitive transformations:

Observation 1 Every primitive transformation is *well-defined i.e.* when applied to any model it yields either ϕ or another model.

Observation 2 For all input models with the same schema, the models output by a primitive transformation also all have the same schema. We denote by $Schema(t, S)$ the schema that results by applying the primitive transformation t to any model of S.

Definition 7 We say that the application of a primitive transformation to a model is *successful* if it does not return ϕ. We say that a primitive transformation t is *schema-dependent (s-d) w.r.t. a schema S* if t is successful for all models of S, otherwise t is *instance-dependent (i-d) w.r.t. S*.

Observation 3 If a primitive transformation t is s-d w.r.t S, then $Inst(S) \subseteq Inst(Schema(t, S))$.

Observation 4 If a primitive transformation t is i-d w.r.t S, then $Inst(S, f) \subseteq$Inst(Schema(t,S)) where f is the proviso associated with t. From Definitions 6 and 11 we see that this proviso is independent of the target model and is thus vacuously true for it. Hence $Inst(S, f) \subseteq Inst(Schema(t, S), f)$.

†This proviso is undecidable in general. However, in practice the schema designer will be using del_E in situations that the proviso is easily verifiable.

The primitive transformations above arise from the structure of an ER schema and are *syntactically complete*, in the sense that without the associated provisos they would allow us to transform any ER schema into any other ER schema. With the addition of the provisos, the transformations become *semantically sound* i.e they result in an ER model with the same instance component as the original model.

The following proposition states that any successful primitive transformation can always be reversed by another primitive transformation:

Proposition 1 For every primitive transformation t such that $t(\langle S, I, Ext_{S,I}\rangle) \neq \phi$ there exists a *reverse primitive transformation* t^{-1} such that $t^{-1}(t(\langle S, I, Ext_{S,I}\rangle)) = \langle S, I, Ext_{S,I}\rangle$.

The proof of this proposition is given in Appendix B and it lists the reverse primitive transformation corresponding to each of the primitive transformations of Definition 6.

From Observation 3 and Definition 4 we have the following:

Observation 5 If for two schemas S and S' there exist primitive transformations t and t' that are s-d w.r.t. S and S', respectively, such that $S' = Schema(t, S)$ and $S = Schema(t', S')$, then S and S' are u-equivalent.

An example is *homonym removal* whereby if n' is not the name of an entity, attribute or relationship in S, then we can rename an entity, attribute or relationship, n, in S by applying the transformation $rename_{E/A/R}\ \langle n, n'\rangle$. The reverse transformation is $rename_{E/A/R}\ \langle n', n\rangle$. Thus, the original and transformed schemas are u-equivalent.

From Observation 4 and Definition 5 we have an analogous result for i-d transformations:

Observation 6 If for two schemas S and S' there exist primitive transformations t and t' that are i-d w.r.t S and S' with associated provisos f and f', and:

- $S' = Schema(t, S)$,
- $S = Schema(t', S')$,
- f' is true for $t(\langle S, I, Ext_{S,I}\rangle)$ whenever f is true for $\langle S, I, Ext_{S,I}\rangle$, and
- f is true for $t'(\langle S', I, Ext_{S',I}\rangle)$ whenever f' is true for $\langle S', I, Ext_{S',I}\rangle$

then S and S' are c-equivalent w.r.t. $f \wedge f'$.

An example is *subtype relationship introduction* whereby if S contains two entities e, e' then we may introduce a subtype relationship by means of the transformation $add_I\ \langle e', e\rangle$ provided $Ext_{S,I}(e') \subseteq Ext_{S,I}(e)$. The reverse transformation $del_I\ \langle e', e\rangle$ will always hold. So we have S and the transformed schema being c-equivalent w.r.t the condition $Ext_{S,I}(e') \subseteq Ext_{S,I}(e)$.

3.2. Composite Transformations

We now generalise the above treatment to sequences of primitive transformations:

Definition 8 A *(composite) transformation* is a sequence of primitive transformations,

$$T = op_1\ arg_1;\ op_2\ arg_2;\ \ldots; op_n\ arg_n$$

where each op_i is one of the primitive transformations of Definition 6 and each arg_i an argument of the appropriate type. The result of applying such a transformation to a model m is given by

$$T(m) = op_n\ arg_n\ (\ldots(op_2\ arg_2\ (op_1\ arg_1\ m))\ldots)$$

From Observations 1 and 2 it follows that any transformation T is well-defined, and that for input models with the same schema its output models also all have the same schema. We denote by $Schema(T, S)$ the schema that results by applying T to any model of S.

Definition 9 Consider a transformation $T = op_1\ arg_1;\ op_2\ arg_2;\ \ldots; op_n\ arg_n$ with provisos $f_1, \ldots, f_n$ associated with the primitive transformations $op_1\ arg_1,\ op_2\ arg_2,\ \ldots, op_n\ arg_n$ respectively. Then for any model m, the *proviso*, f, associated with T is that:

$$f_i \text{ holds for } op_{i-1}\ arg_{i-1}\ (\ldots(op_2\ arg_2\ (op_1\ arg_1\ m))\ldots) \text{ for all } 1 \leq i \leq n$$

If f holds for all models of S, then T is successful for all models of S and we say that T is *schema-dependent (s-d) w.r.t. S*. Otherwise T is *instance-dependent (i-d) w.r.t. S*.

Regarding the reversibility of transformations, from Proposition 1 it follows that for every transformation $T = op_1\ arg_1;\ op_2\ arg_2;\ \ldots; op_n\ arg_n$ such that $T(m) \neq \phi$ there exists a transformation T^{-1} such that $T^{-1}(T(m)) = m$: let $T^{-1} = op'_n\ arg'_n;\ op'_{n-1}\ arg'_{n-1};\ \ldots; op'_1\ arg'_1$ where each $op'_i\ arg'_i$ is the reverse primitive transformation of $op_i\ arg_i$.

Finally, Observations 3 and 4 generalise directly to composite transformations, as do Observations 5 and 6. For example, the entity/attribute and entity/relationship equivalences of Figure 1 illustrate two pairs of composite transformations between u-equivalent schemas while the introduction of union entity and introduction of intersection entity equivalences of Figure 2 illustrate two pairs of composite transformations between c-equivalent schemas:

- *Entity/Attribute Equivalence.* If S contains an attribute a, an entity e, and an association between them $\langle Null, e, a, c_1, c_2\rangle$, and S does not contain an entity e_a, then the following sequence of s-d primitive transformations give a composite s-d transformation that converts the attribute a to a new entity e_a:

$$add_E\ \langle e_a, a\rangle;\ add_R\ \langle r, e, e_a, c_1, c_2, \langle Null, e, a\rangle\rangle;\ del_A\ \langle e, a\rangle$$

  Conversely, if S' contains an entity e_a whose only association is $\langle r, e, e_a, c_1, c_2\rangle$, and S' does not contain an attribute a, then the following sequence of s-d primitive transformations give a composite s-d transformation that converts the entity e_a to a new attribute a of e:

$$add_A\ \langle e, a, c_1, c_2, e_a, \langle r, e, e_a\rangle\rangle;\ del_R\ \langle r, e, e_a\rangle;\ del_E\ e_a$$

  Thus, S and S' are u-equivalent.

- *Entity/Relationship Equivalence.* If S contains an entity e_R for which the only associations are $\langle r_1, e_1, e_R, c_1, 1{:}1\rangle$ and $\langle r_2, e_R, e_2, 1{:}1, c_2\rangle$, and S does not contain the relationship r, then the following sequence of s-d primitive transformations transforms e_R into a new relationship r:

$$\begin{gathered} add_R\ \langle r, e_1, e_2, c_1, c_2, \\ \{\langle e_1(s_1), e_R(s_1), e_2(s_2)\rangle \mid s_1 \in \langle r_1, e_1, e_R\rangle \wedge s_2 \in \langle r_2, e_R, e_2\rangle \wedge e_R(s_1) = e_R(s_2)\}\rangle; \\ del_R\ \langle r_1, e_1, e_R\rangle;\ del_R\ \langle r_2, e_R, e_2\rangle;\ del_E\ e_R \end{gathered}$$

  Conversely, if S' contains a relationship $\langle r, e_1, e_2, c_1, c_2\rangle$ and does not contain the entity e_R or the relationships r_1, r_2, then the following sequence of s-d primitive transformations transforms r into a new entity e_R:

$$\begin{gathered} add_E\ \langle e_R, \langle r, e_1, e_2\rangle\rangle;\ add_R\ \langle r_1, e_1, e_R, c_1, 1{:}1, \langle r, e_1, e_2\rangle\rangle; \\ add_R\ \langle r_2, e_R, e_2, 1{:}1, c_2, \langle r, e_1, e_2\rangle\rangle;\ del_R\ \langle r, e_1, e_2\rangle \end{gathered}$$

  Thus, S and S' are u-equival.

- *Introduction of a Union Entity.* If S contains two entities e_1, e_2 and does not contain the entity e, then the following sequence of s-d primitive transformations creates a new entity e as the union of e_1 and e_2:

$$add_E\ \langle e, e_1 \cup e_2\rangle;\ add_I\ \langle e_1, e\rangle;\ add_I\ \langle e_2, e\rangle$$

Conversely, if S' contains entities e, e_1, e_2 and subtype relationships $\langle e_1, e\rangle, \langle e_2, e\rangle$, and e participates in no associations in S' then, provided $Ext_{S',I}(e) = Ext_{S',I}(e_1) \cup Ext_{S',I}(e_2)$, we can apply to following sequence of primitive transformations, to obtain S:

$$del_I\ \langle e_2, e\rangle;\ del_I\ \langle e_1, e\rangle;\ del_E\ e$$

Thus S and S' are c-equivalent w.r.t the above proviso.

- *Introduction of an Intersection Entity.* Similar remarks apply to the introduction of an intersection entity:
$$add_E\ \langle e, e_1 \cap e_2\rangle;\ add_I\ \langle e, e_1\rangle;\ add_I\ \langle e, e_2\rangle$$
and its reverse transformation:
$$del_I\ \langle e, e_2\rangle;\ del_I\ \langle e, e_1\rangle;\ del_E\ e$$

3.3. *Knowledge-Based Transformations*

We can enhance the expressiveness of primitive transformations by allowing them to take an extra parameter. This will encode a user-defined condition on the instance component of the model which must be satisfied in order for the transformation to be successful. We call such transformations *knowledge-based* (k-b) ones in order to distinguish them from the family of two-parameter primitive transformations considered above. However, semantically there is no difference between i-d and k-b transformations since both require instances to satisfy a condition.

Definition 10 For each of the primitive transformations of Definition 6 there is a corresponding *knowledge-based* version that takes one extra parameter of type $Models \rightarrow Bool$. This is a condition to be evaluated on the model parameter, and is expressed in the mapping language L. The same name is used for the two-parameter and the three-parameter versions of the primitive transformations — the number of arguments passed to the transformations distinguishes which version is meant. Each three-parameter transformation, *op*, is defined in terms of the two-parameter transformation as follows:

$$\begin{aligned} op\ arg\ f\ m &= op\ arg\ m, \text{ if } f(m) \\ &= \phi, \text{ otherwise} \end{aligned}$$

Regarding the reversibility of k-b transformations, consider a successful k-b primitive transformation:

$$op\ arg\ f\ \langle S, I, Ext_{S,I}\rangle = \langle S', I, Ext_{S',I}\rangle$$

Since $op\ arg\ \langle S, I, Ext_{S,I}\rangle = \langle S', I, Ext_{S',I}\rangle$, by Proposition 1 there is a primitive transformation $op'\ arg'\ \langle S', I, Ext_{S',I}\rangle$ which returns $\langle S, I, Ext_{S,I}\rangle$. So if there exists a condition f' that is true for $\langle S', I, Ext_{S',I}\rangle$ whenever f is true for $\langle S, I, Ext_{S,I}\rangle$, then we can reverse the k-b primitive transformation by $op'\ arg'\ f'\ \langle S', I, Ext_{S',I}\rangle$. Examples of such pairs of k-b primitive transformations are the synonym removal equivalences discussed in Section 4 below. The treatment generalises to k-b composite transformations such as:

- *Attribute/Subtype Equivalence.* We can define a pair of transformations that implements the equivalence in Example 2. In particular, if S contains an attribute a, an entity e, and an association between them $\langle Null, e, a, m_1 : n_1, m_2 : n_2\rangle$, we may use knowledge about the domain of a to create a number of sub-entity types for e, and remove a:

$$\begin{gathered} add_E\ \langle e_1, \{e(s) \mid s \in \langle Null, e, a\rangle \wedge a(s) = v_1\}\rangle;\ \ldots; \\ add_E\ \langle e_n, \{e(s) \mid s \in \langle Null, e, a\rangle \wedge a(s) = v_n\}\rangle; \\ add_I\ \langle e_1, e\rangle;\ \ldots;\ add_I\ \langle e_n, e\rangle;\ del_A\ \langle e, a\rangle\ f \end{gathered}$$

where the condition f is as defined in Example 2.

Conversely, if S' contains entity types $e, e_1, \ldots, e_n$ such that $e_1, \ldots, e_n$ are subtypes of e and participate in no associations, and also S' does not contain an attribute a, then we can replace $e_1, \ldots, e_n$ by a new attribute a of e, where the condition f' is as defined in Example 2:

$$add_A\ \langle e, a, m_1 : n_1, m_2 : n_2, \{v_1, \ldots, v_n\}, \cup_{i=1\ldots n}\{\langle x, v_i\rangle \mid x \in e_i\}\rangle\ f';$$
$$del_I\ \langle e_n, e\rangle;\ \ldots;\ del_I\ \langle e_1, e\rangle;\ del_E\ e_n;\ \ldots del_E\ e_1$$

Thus, S and S' are c-equivalent w.r.t. the condition $f \wedge f'$.

Further examples are the four k-b composite transformations below which are all easily seen to be reversible by s-d transformations and thus give rise to c-equivalent schemas with respect to the stated provisos:

- *Redundant Attribute Removal.* If the extent of an association between a sub-entity e' and an attribute a is subsumed by the extent of the association between a super-entity e and a then a can be removed from e':

$$del_A\ \langle e', a\rangle\ (Ext_{S,I}(\langle Null, e', a\rangle) = \{s \mid s \in Ext_{S,I}(\langle Null, e, a\rangle) \wedge e(s) \in Ext_{S,I}(e')\})$$

- *Optional Attribute Removal.* An attribute a with cardinality $\langle 0 : x, c\rangle$ may be removed from an entity e to a set of its subtypes $e_1, \ldots e_n$ if a is only associated with these subtypes and it is a mandatory attribute for these types:

$$add_A\ \langle e_1, a, 1 : x, c, a, \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_1\}\rangle$$
$$(Ext_{S,I}(e_1) = \{e(s) \mid s \in Ext_{S,I}(\langle Null, e, a\rangle) \wedge e(s) \in Ext_{S,I}(e_1)\});$$
$$\vdots$$
$$add_A\ \langle e_n, a, 1 : x, c, a, \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_n\}\rangle$$
$$(Ext_{S,I}(e_n) = \{e(s) \mid s \in Ext_{S,I}(\langle Null, e, a\rangle) \wedge e(s) \in Ext_{S,I}(e_n)\});$$
$$del_A\ \langle e, a\rangle\ (\{e(s) \mid s \in Ext_{S,I}(\langle Null, e, a\rangle\} = \cup_{i=1\ldots n} Ext_{S,I}(e_i))$$

- *Generalisation of Attributes.* If a set of entity types $e_1, \ldots e_n$ have a common super-entity e and their extents cover the extent of e then an attribute a common to all sub-entities may be moved to e:

$$add_A\ \langle e, a, c_1, c_2, a, \cup_{i=1\ldots n}\langle Null, e_i, a\rangle\rangle\ (\cup_{i=1\ldots n} Ext_{S,I}(e_i) = Ext_{S,I}(e));$$
$$del_A\ \langle e_1, a\rangle;\ \ldots;\ del_A\ \langle e_n, a\rangle$$

  The values c_1, c_2 in the above transformation may be computed either from the constraints on the sub-types or from the extent information.

- *Redundant Relationship Removal.* As illustrated in Figure 3(d):

$$del_R\ \langle r, e'_1, e'_2\rangle$$
$$(Ext_{S,I}(\langle r, e'_1, e'_2\rangle) = \{s \mid s \in Ext_{S,I}(\langle r, e_1, e_2\rangle) \wedge e_1(s) \in Ext_{S,I}(e'_1) \wedge e_2(s) \in Ext_{S,I}(e'_2)\})$$

  and Figure 3(e):

$$del_R\ \langle r, e'_1, e_2\rangle\ (Ext_{S,I}(\langle r, e'_1, e_2\rangle) = \{s \mid s \in Ext_{S,I}(\langle r, e_1, e_2\rangle) \wedge e_1(s) \in Ext_{S,I}(e'_1)\})$$

4. THE SCHEMA INTEGRATION PROCESS

We are now ready to examine how the formalism developed so far can be used to underpin the schema integration process. We are interested in giving a formal foundation for the transformations that any methodology may perform, and we do not propose a particular methodology, of which many already exist [5, 12, 16, 21, 18]. For any particular methodology, a set of composite transformation rules will be defined using the primitive transformations described in the previous section. The types of transformations that should be applied depends on the kind of equivalence (transformational, mapping or behavioural) that is to be maintained. Dividing the task into the standard series of phases already mentioned in the introduction [4] and illustrated in Figures 1–3, the broad steps involved are outlined below. However, it should be noted that as far as our notion of schema transformation is concerned there is really no distinction between phases 2–4.

1. *Rre-Integration*: The individual models, $\langle S_1, I_1, Ext_{S_1,I_1}\rangle, \ldots, \langle S_n, I_n, Ext_{S_n,I_n}\rangle$, are prepared for integration by transforming each of them into the ER CDM. Their *union model* is then formed. This is defined as

$$\langle S_1 + \ldots + S_n, I_1 \cup \ldots \cup I_n, Ext_{S_1,I_1} + \ldots + Ext_{S_n,I_n}\rangle$$

where for any schemas $S_1 = \langle Ents_1, Atts_1, Incs_1, Assocs_1\rangle, \ldots, S_n = \langle Ents_n, Atts_n, Incs_n, Assocs_n\rangle$:

$$S_1 + \ldots + S_n = \langle \cup_{i=1\ldots n} Ents_i, \cup_{i=1\ldots n} Atts_i, \cup_{i=1\ldots n} Incs_i, \cup_{i=1\ldots n} Assocs_i\rangle$$

and for any functions $f_1, \ldots, f_n$ with disjoint domains $D_1, \ldots, D_n$ and, not necessarily disjoint, ranges $R_1, \ldots, R_n$:

$$\begin{array}{rcl} f_1 + \ldots + f_n & : & (D_1 \cup \ldots \cup D_n) \to (R_1 \cup \ldots \cup R_n) \\ (f_1 + \ldots + f_n)(d) & = & f_i(d) \text{ for } d \in D_i \end{array}$$

We notice that the union model is indeed a model and that name ambiguities are avoided since the schema identifier component of a name is unique to each schema.

2. *Schema Conforming*: We next need to resolve conflicts between the participating models of the union model by applying homonym- and synonym-removing transformations. The order in which the participating models are examined will depend on the particular integration strategy *e.g.* one-shot, ladder, balanced *etc* [4] and further consideration of this is beyond the scope of this paper.

   There will be an *entity overlap* between a pair of participating models $\langle S, I, Ext_{S,I}\rangle$ and $\langle S', I', Ext_{S',I'}\rangle$ if there are entity types e of S and e' of S' such that $Item(e) = Item(e')$; there will be an entity *conflict* if $Ext_{S,I}(e) \neq Ext_{S',I'}(e')$.

   In the case of an entity conflict involving e and e', we can change e, say, to a new name e'' which has $Schema(e'') = S$ by applying the homonym removal operation $rename_E\ \langle e, e''\rangle$. Conversely, in the case that e and e' do not overlap but have the same extent, we can apply the following *entity synonym removal* operation to change the item identifier of e to be the same as that of e' (where $||$ denotes concatenation):

$$rename_E\ \langle e, Schema(e)||Item(e')\rangle\ (Ext_{S,I}(e) = Ext_{S,I}(e'))$$

   Since this k-b primitive transformation is reversible by the s-d transformation $rename_E$ $\langle Schema(e)||Item(e'), e\rangle$, the original and transformed schemas are c-equivalent.

   Similarly, there will be a *relationship overlap* between a pair of participating models $\langle S, I, Ext_{S,I}\rangle$ and $\langle S', I', Ext_{S',I'}\rangle$ if there are schemes $\langle r, e_1, e_2\rangle$ in S and $\langle r', e_1', e_2'\rangle$ in S' such that $Item(e_1) = Item(e_1')$, $Item(e_2) = Item(e_2')$ and $Item(r) = Item(r')$; if $Ext_{S,I}(\langle r, e_1, e_2\rangle)$ $\neq Ext_{S',I'}(\langle r', e_1', e_2'\rangle)$ then there will be a relationship *conflict* between $\langle S, I, Ext_{S,I}\rangle$ and $\langle S', I', Ext_{S',I'}\rangle$.

   In the case of a relationship conflict involving r and r', we can change r, say, to a new name r'' which has $Schema(r'') = S$ by applying the homonym removal operation $rename_R\ \langle r, r''\rangle$.

Conversely, in the case that r and r' do not overlap but have the same extent and relate pairs of entity types with the same item identifiers we can use the following *relationship synonym removal* operation to change the item identifier of r to be the same as that of r':

$$\begin{gathered} rename_R\ \langle r, Schema(r)||Item(r')\rangle \\ (\exists e_1, e_2, e_1', e_2'.\langle r, e_1, e_2\rangle \in Schemes \wedge \langle r', e_1', e_2'\rangle \in Schemes \wedge \\ Ext_{S,I}(\langle r, e_1, e_2\rangle) = Ext_{S,I}(\langle r', e_1', e_2'\rangle) \wedge Item(e_1) = Item(e_1') \wedge Item(e_2) = Item(e_2')) \end{gathered}$$

Since this k-b primitive transformation is reversible by the s-d transformation $rename_R$ $\langle Schema(r)||Item(r'), r\rangle$ the original and transformed schemas are c-equivalent.

Finally, there will be an *attribute overlap* between a pair of models $\langle S, I, Ext_{S,I}\rangle$ and $\langle S', I', Ext_{S',I'}\rangle$ if there are attributes a of S and a' of S' such that $Item(a) = Item(a')$; there will be an attribute *conflict* if either

(i) $Ext_{S,I}(a) \neq Ext_{S',I'}(a')$, or

(ii) there are schemes $\langle Null, e, a\rangle$ in S and $\langle Null, e', a'\rangle$ in S' such that $Item(e) = Item(e')$ and $Ext_{S,I}(\langle Null, e, a\rangle) \neq Ext_{S',I'}(\langle Null, e', a'\rangle)$.

In the case of an attribute conflict involving a and a', we can change a, say, to a new name a'' which has $Schema(a'') = S$ by applying the homonym removal operation $rename_A\ \langle a, a''\rangle$. Conversely, in the case that a and a' do not overlap but have the same extent, are associated with the same set of entity types and have the same extents for these associations we can use the following *attribute synonym removal* operation to change the item identifier of a to be the same as that of a':

$$\begin{gathered} rename_A\ \langle a, Schema(a)||Item(a')\rangle\ (Ext_{S,I}(a) = Ext_{S,I}(a') \wedge \\ \{Item(e) \mid \langle Null, e, a\rangle \in Schemes\} = \{Item(e) \mid \langle Null, e, a'\rangle \in Schemes\} \wedge \\ \forall e, e' \in Ents.((\langle Null, e, a\rangle \in Schemes \wedge \langle Null, e', a'\rangle \in Schemes \wedge Item(e) = Item(e')) \\ \Rightarrow (Ext_{S,I}(\langle Null, e, a\rangle) = Ext_{S,I}(\langle Null, e', a'\rangle)))) \end{gathered}$$

Again, this k-b primitive transformation is reversible by the s-d transformation $rename_A$ $\langle Schema(a)||Item(a'), a\rangle$ and the original and transformed schemas are c-equivalent.

3. *Schema Merging*: Provided that all conflicts have been resolved, we can undertake a renaming operation for each each entity, attribute and relationship name n in the union schema whereby the schema identifier of n is replaced by a new schema identifier, SID, which uniquely identifies the union schema:

$$rename_X\ \langle n, SID||Item(n)\rangle$$

We notice that the original and transformed schemas are c-equivalent w.r.t. the above proviso.

A side effect of this renaming operation is the implicit merging of entities and attributes which had the same item name in the union model, and similarly the implicit merging of relationships that had the same item name and were between entities with the same item name. So this process subsumes the **entity merge** transformation illustrated in Figure 2(a).

Transformations to add implied unions, intersections, relationships and subtype relationships can then be applied.

4. *Schema Restructuring*: This will finally involve using transformations which eliminate redundant structures and improve the quality of the unified model, as illustrated in Figure 3.

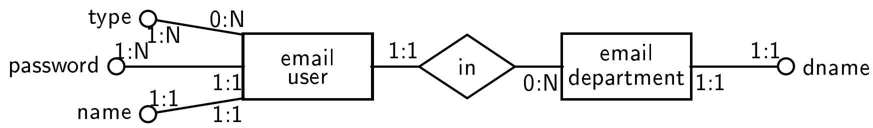
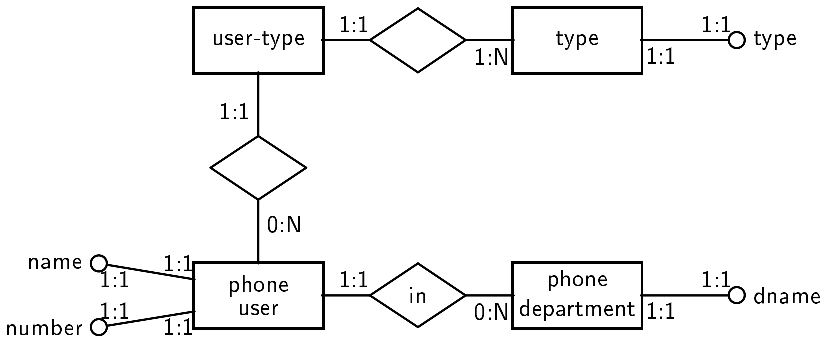
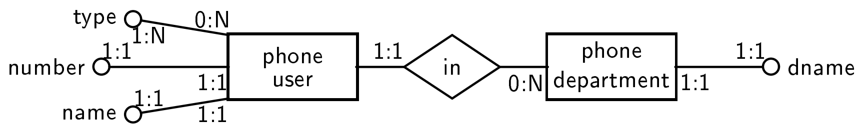


Fig. 4: Email Database Schema



(a) Partially Conformed



(b) Fully Conformed

Fig. 5: Telephone Database Schema

5. A CASE STUDY

To demonstrate the practical use of our formalism, in this section we show how to define one new composite s-d transformation and one new k-b transformation. These examples have arisen from the use of our formalism for schema integration at two large organisations, as part of the Esprit Framework IV HOD project (No. EP20620). Simplified fragments of the two databases to be integrated are shown in Figures 4 and 5(a), which are shown in a state after a certain amount of schema conforming. This identified that phone user and email user are overlapping entities, as are phone department and email department.

The type attribute has the same domain in both schemas, but is attached to different entities, Analysis showed that we should in the Telephone schema move the type attribute from the type entity onto the phone user entity, since the phone user was found to overlap the email user entity to which type is attached in the Email schema. This may by performed by two s-d transformations. Firstly an *entity/relationship equivalence*, replacing user-type by a many-many relationship between phone user and the type entity. Secondly, a new *entity-attribute/attribute equivalence*, which transforms the entity and attribute pair — both called type — into a single attribute. The result of using this transformation is shown in Figure 5(b), which is a fully conformed version of the Telephone database schema. We may now merge the two schemas by adding union entities user and department, to give the schema shown in Figure 6.

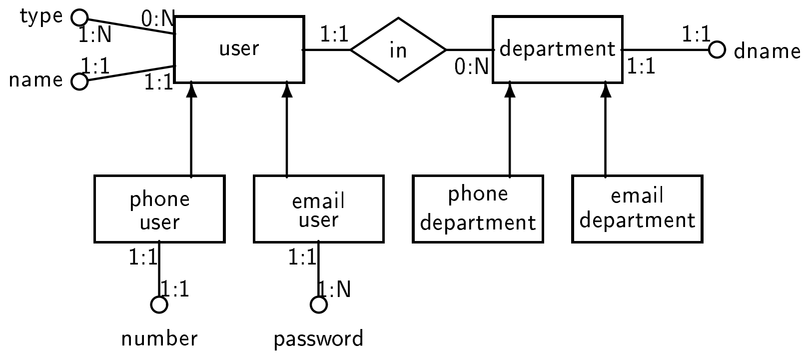


Fig. 6: Email and Telephone Databases Merged

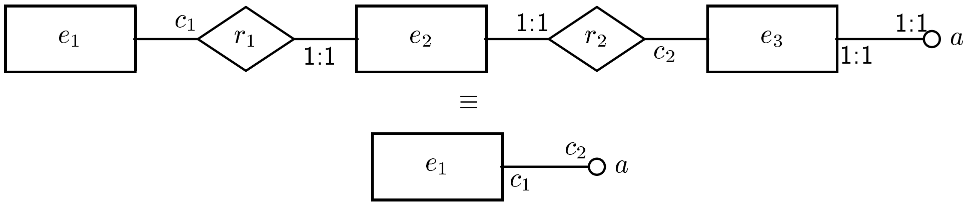
This sequence of transformations was found to be a very common one, making it convenient for us to define a new s-d transformation of *many-valued-attribute/entity equivalence* illustrated in Figure 7(a), which shows that an entity e_3 with a single key attribute a and in a relationship with e_2, which itself is related to e_1, can be replaced by a, regardless of the schema instance, using the transformation:

$$add_R\ \langle Null, e_1, a, c_1, c_2, \{\langle e_1(s_1), e_3(s_3)\rangle \mid s_1 \in Ext_{S,I}(\langle r_1, e_1, e_2\rangle) \wedge s_2 \in Ext_{S,I}(\langle r_2, e_2, e_3\rangle) \wedge$$
$$e_2(s_1) = e_2(s_2) \wedge s_3 \in Ext_{S,I}(\langle Null, e_3, a\rangle) \wedge e_3(s_2) = e_3(s_3)\}\rangle;$$
$$del_R\ \langle r_1, e_1, e_2\rangle;\ del_R\ \langle r_2, e_2, e_3\rangle;\ del_E\ e_2;\ del_E\ e_3$$

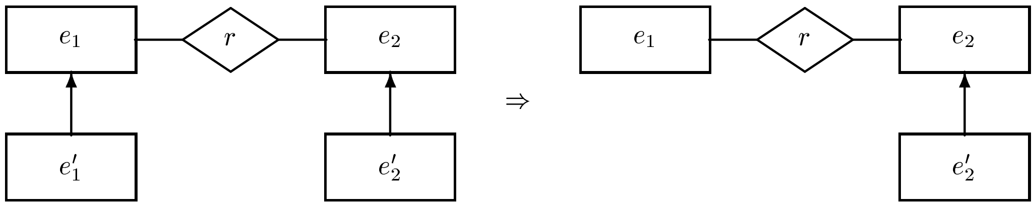
The merged schema in Figure 6 contains two entities — phone department and email department — which do not have attributes, which normally indicates that the property they represent should not be represented by an entity symbol. In fact, these two entities are just a vestige of entities in the separate databases and can be derived from the association of a department with user of subtype phone user or email user. However, the current schema allows departments to be denoted as phone department or email department independently of a user's subclass. Thus we cannot apply an s-d transformation to remove phone department and email department, but we can use a k-b transformation dependent on each phone department being related via the relationship in with a phone user, and each email department being similarly related to an email user. This can be generalised to the k-b *redundant subtype removal* transformation illustrated in Figure 7(b), which removes subtype e_1' of e_1 when e_1 is related to e_2 and only instances of e_1' are related to instances of the subtype e_2' of e_2:

$$del_I\ \langle e_1, e_1'\rangle\ (Ext_{S,I}(e_1') = \{s \mid s \in Ext_{S,I}(e_1) \wedge t \in Ext_{S,I}(\langle r, e_1, e_2\rangle) \wedge$$
$$e_1(t) = s \wedge e_2(t) \in Ext_{S,I}(e_2')\});$$
$$del_E\ e_1'$$

During the HOD project, it became apparent that a relatively small number of transformations were being used repeatedly during the schema integration process. This is not a surprising result since a well-designed database schema will have been built using a certain design 'style', and thus when we merge two such schemas we should expect to have to transform all applications of a particular aspect of one design style to match the equivalent aspect of the other style. For example, one schema may tend to use subclasses where another would use an attribute, and thus we would need multiple applications of the attribute/subtype equivalence.



(a) Entity/Many-Valued Attribute Equivalence



(b) Redundant Subtype Removal

Fig. 7: Two New Transformations Introduced for the Case Study

Most of the transformations required during HOD were of the type that we have summarised in Figures 1–3, and the only two new transformations required were the two described above. Thus, if a schema integration tool were to support the definition of new composite transformations from the set of primitive transformations, the resulting new transformations would be likely to used many times during a real schema integration exercise, justifying the time spent in developing the new transformations.

6. HANDLING RICHER MODELLING CONSTRUCTS

Our choice of a binary ER model for the CDM was made in order not to overly complicate the formalisation of schema equivalence. There are two ways in which richer modelling constructs such as n-ary relationships, complex attributes and generalisation hierarchies can be handled: (a) by extending the formal framework to include these as primitive CDM constructs, or (b) by transforming these constructs into the present CDM during the pre-integration phase and subsequently using any extra knowledge to underpin k-b transformations. For approach (a), the definition of schemas and instances would have to be suitably extended, as would the set of primitive transformations; we give details of this approach in a forthcoming paper [13]. For approach (b), the following transformations would occur during the pre-integration phase:

- An *n-ary relationship* r between n entities $e_1, \ldots, e_n$ can be transformed into a new entity r with n binary relationships between r and $e_1, \ldots, e_n$.
- A *complex attribute* a of an entity e which is an aggregation of n attributes $s_1, \ldots, s_n$, e.g. as used by [20], can be transformed into an entity a with n attributes $s_1, \ldots, s_n$ and a relationship r between e and a. The domain of a is the Cartesian product of the domains of $s_1, \ldots, s_n$ and this knowledge can subsequently be used in applying k-b transformations.
- A *generalisation hierarchy* g between an entity e and disjoint subtypes $e_1, \ldots, e_n$, e.g. as used by [3], can be transformed into a subtype relationship between e and each of $e_1, \ldots, e_n$. Again, the knowledge that the subtypes $e_1, \ldots, e_n$ form a partition of e can be used in applying k-b transformations.

7. COMPARISON WITH RELATED WORK

7.1. Previous Notions of Schema Equivalence

We defined the previous notions of transformational, mapping and behavioural equivalence in Section 1.2 above. Our definition of u-equivalence provides a semantic foundation for transformational equivalence in the sense that any transformation of one schema into another is semantically sound provided the two schemas are u-equivalent. Mapping equivalence is also a special case of our definitions of u-equivalence (for condition-less mapping equivalences) or c-equivalence (for conditional mapping equivalences) in that the mappings between $Range(Ext_{S,I})$ and I and $Range(Ext_{S',I})$ and I must induce a one-to-one correspondence between $Range(Ext_{S,I})$ and $Range(Ext_{S',I})$. Finally, our definition of u-equivalence also subsumes the notion of behavioural equivalence, as follows. Consider two schemas S and S' that are u-equivalent according to Definition 4. Let

$$\begin{aligned} Ents \cup Atts \cup Schemes &= \{o_1, \ldots, o_n\} \\ Ents' \cup Atts' \cup Schemes' &= \{p_1, \ldots, p_m\} \\ I &= \{i_1, \ldots, i_r\} \end{aligned}$$

We recall from Section 3.1 that a query Q over the model $\langle S, I, Ext_{S,I}\rangle$ is an expression in L whose variables are a subset of $Ents \cup Atts \cup Schemes$. The value of Q is given by

$$Q[o_1/Ext_{S,I}(o_1), \ldots, o_n/Ext_{S,I}(o_n)]$$

Now each $Ext_{S,I}(o_i) = expr_{o_i}$ for some expression $expr_{o_i}$ in L over I. Thus, the value of Q is

$$Q[o_1/expr_{o_1}, \ldots, o_n/expr_{o_n}]$$

Moreover, each $i \in I$ is equal to $expr_i$ for some expression $expr_i$ in L over $Range_{S',I}$. Thus the value of Q is

$$Q[o_1/expr_{o_1}, \ldots, o_n/expr_{o_n}][i_1/expr_{i_1}, \ldots, i_r/expr_{i_r}].$$

But this is just the value of the query

$$Q[o_1/expr_{o_1}, \ldots, o_n/expr_{o_n}][i_1/expr_{i_1}, \ldots, i_r/expr_{i_r}][Ext_{S',I}(p_1)/p_1, \ldots, Ext_{S',I}(p_m)/p_m]$$

over the model $\langle S', I, Ext_{S',I}\rangle$. This argument also applies in the case that the schemas S and S' are c-equivalent with respect to a condition f, in which case the instance I above satisfies f.

Table 2 in Section 1.2 classified the transformations of Figures 1–3 according to whether they obey transformational, mapping or behavioural equivalence. In previous sections of the paper we have shown how all of these transformations can be expressed using in our formalism. In practice, schema conformance transformations are applied bi-directionally (apart from synonym- and homonym-removal) while schema merging and restructuring ones are applied uni-directionally. For each transformation, it is easy to show formally that the properties of transformational, mapping or behavioural equivalence shown for it in Table 2 indeed hold, given our discussion above of how these notions relate to our definitions of u-equivalence or c-equivalence.

Table 3 categorises these transformations according to whether they are schema-dependent (s-d), instance-dependent (i-d) or knowledge-based (k-b). We observe that our approach clearly differentiates the type of equivalence that each transformation preserves since each transformation is classified into precisely one category, as opposed to possibly several categories as in Table 2. Table 3 categorises the left-to-right transformation under the column marked $\rightarrow$ and the right-to-left transformation under $\leftarrow$.

7.2. Previous Work on Formalising Schema Equivalence

A considerable amount of work has been conducted into the notion of equivalence between relational and other schemas. For example, the work of [17, 2, 9, 6] defines various notions of

Integration Stage	Transformation Description	$\rightarrow$	$\leftarrow$
Schema conform (Figure 1)	Entity/attribute	s-d	s-d
	Entity/relationship	s-d	s-d
	Attribute/is-a	k-b	k-b
Schema merge (Figure 2)	Entity merge	k-b	s-d
	is-a introduction	i-d	s-d
	Addition of union	s-d	k-b
	Addition of intersection	s-d	k-b
Schema restructuring (Figure 3)	Redundant attribute removal	k-b	s-d
	Optional attribute removal	k-b	s-d
	Generalisation of attributes	k-b	s-d
	Relationship removal (1)	k-b	s-d
	Relationship removal (2)	k-b	s-d

Table 3: A New Classification of Common Schema Transformations

u-equivalence for relational models, [14, 15, 18] u-equivalence for ER models, and [8] u-equivalence for a more general model based on partially ordered sets. None of this work provides a mechanism for defining c-equivalence, since it is always necessary in these approaches for the schemas to be equivalent for all possible instances. In effect, it is assumed the types of the schemas are constrained to make them u-equivalent, which we argue limits the practical application of the mappings in real database systems which will lack such specific typing of domains. This may be illustrated by reviewing Example 2, for which the above approaches would require that the domain of attribute a be limited to n values if the equivalence is to hold. By contrast, our approach allows for a k-b transformation of an a with any domain into $e_1, \ldots, e_n$, with the condition that the instance of a it is actually applied to contains at most n values.

Of the previous work mentioned above, [14, 15] is closest to ours, and so we will study it in detail here to clarify the distinction and emphasise the consequences. In [14] a formal approach to schema equivalence was proposed, based on the notion of the *information capacity* of a schema. In particular, a schema S_1 was said to *dominate* a schema S_2, denoted by $S_1 \geq S_2$, if all possible instances of S_2 could be mapped to instances of S_1, and two schemas S_1, S_2 were said to be equivalent if each dominates the other. This notion of equivalence corresponds to our requirement that there exists a pair of s-d transformations between S_1 and S_2. The notion of *total* dominance of [14], denoted by $S_1 > S_2$, can be characterised in our formalism as stating that there exists an s-d transformation T such that $S_1 = \mathit{Schema}(T, S_2)$, and that all possible transformations T^{-1} such that $S_2 = \mathit{Schema}(T^{-1}, S_1)$ are either i-d or k-b.

A simple example of such a transformation pair can be found in Figure 3(b) where the left-hand schema, S_1, has total dominance over the right-hand schema, S_2, since S_1 allows members of e_1 which are not members of e_2 to be associated with a, whilst S_2 does not. The schema dominance approach of [14] can transform S_2 to S_1, but not vice versa. In contrast, with our approach we can perform both transformations, the S_1 to S_2 transformation being an k-b one requiring that members of a be associated only with members of e_2.

A more complex example of such a transformation pair can be found in Figure 1(c). If the association between e and a has cardinality $1:N$ i.e. each instance of e can be associated with an arbitrary number of attribute values, then the left-hand schema, S_1, has total dominance over the right-hand schema, S_2, since n is finite. The schema dominance approach can transform S_2 to S_1, but not vice versa. In contrast, with our approach we can perform both transformations, the S_1 to S_2 transformation requiring that the extent of the association between e and a satisfies the cardinality constraint $1{:}n$ even if the schema is not so restricted.

The approach of [14] was further developed in [15], where it was found that allowing arbitrary functions to define the mapping between schemas leads to the testing of schema dominance being undecidable. It was thus proposed that a set of primitive *structural transformations* be defined which individually may be tested for dominance, and which may be built into sequences which perform useful transformations on the schemas. Although capable of transforming a wide range of schemas, these sequences of structural transformations are able to perform fewer transformations than the transformations that we have defined in Section 3, for the following reasons:

(i) Sequences of structural transformations containing conflicting dominances, such as $S_1 \leq S' \geq S_2$, lead to the relationship between S_1, S_2 being undefined in [15]. In contrast, we can define composite transformations that pass through an intermediate step such as S' which dominates both the initial and final schemas. For example, when we draw an equivalence between an entity and a relationship in Figure 1(b), our formalisation of this equivalence has a stage where both e_r and r exist. This intermediate schema dominates both of the schemas in the equivalence, since e_r and r could have different extents.

(ii) The three structural transformations defined in [15] are limited in what they may achieve. In particular, the only 'entity-creating' transformation is limited to producing an entity with the same domain as an already existing entity. In contrast, we allow the domains of new entities to be defined by any expression over existing entities and relationships expressible in the mapping language. For example, when we draw an equivalence between an attribute and a set of subtypes in Figure 1(c), we can define the extent of e_{v_1} to consist of those members of the extent of e which are associated with the value v_1 of attribute a.

Other work [5, 12, 21, 10] allows for equivalence to be conditional upon the statement of some form of *integration assertions*, which allow links to be made between schemas by making various restrictions on their instances. For example, it is possible in all the cited work to state an integration assertion that the extent of one entity is a subset of another. This gives some of the capabilities of our notion of c-equivalence, but suffers from two limitations. Firstly, it is not clear exactly what set of integration assertions should be provided. For example, it is only [10] that provides an assertion which links the domain of an attribute with the extents of various subclasses, hence achieving the k-b transformation necessary for Example 2. All the other approaches appear to be unable to state this equivalence. Secondly, the fact that the assertions are fixed means that for an equivalence to be drawn between two schemas, it is often the case that a view needs to made on one schema which is union-compatible with that of another schema, in order to allow the assertion that their common items have the same extent to be stated.

Our approach has the advantage that there is never any requirement to devise new assertions (which in any schema integration tool would require some method of defining these assertions) separately from the schema transformation rules. Thus our formalism lends itself to a simple 'programming' approach, with any restriction on the applicability of a rule being built into the condition of that rule. Whilst the approach of using separate assertions might be adapted to allow the definition of new assertions, none of the previous work details how this might be done. In any case, such a technique seems to make the definition of new schema transformation rules more complex than necessary. A similar approach to ours was recently described in [11] where the notion of a database 'context' constrains instances so that schemas can be considered equivalent.

8. CONCLUSIONS

We have proposed a unifying framework for the various methodologies for schema integration, which employ different notions of schema equivalence. We have proposed a new formalisation of the notions of schema, instance, model, and schema equivalence and have shown how the previous notions of transformational, mapping, and behavioural equivalence are variants of our more general definition of equivalence. We have also clarified what kind of schema equivalence the transformations commonly found in the literature obey. Finally, we have defined a semantically sound set of primitive transformations that can be used to express both existing and new schema transformations. Along the way, we have shown how some of the common transformations found in the literature can be defined using our set of primitive transformations.

This work has practical application in the implementation of tools to aid the schema integration process. Building such a tool using our definition of schema equivalence as the basis for applying schema transformations would have several advantages:

- *Data and rules in the database can be used to verify the application of transformations, if necessary.* Our formalism allows users to 'program' new transformations to suit particular environments, using existing primitive and composite transformations. Since the condition on each primitive transformation step is stated, these new transformations will clearly identify which steps are schema-dependent, and thus need not be verified against the database, and which are knowledge-dependent, and hence worth the cost of verifying against the data and/or rules in the database.

- *Transformations may be used to verify data in the database.* In practice, databases rarely contain entirely 'clean' data, and some data cleansing is often required when integrating two databases. We may use the failure of a condition of a transformation to prompt the user with the data which caused the failure. The user could then decide whether the transformation was in fact incorrect, or whether the data causing the failure needs correction.

- *Transformations may be used to specify schema updates.* An update to a schema can be represented as a primitive or composite transformation (possibly a knowledge-dependent one). The reverse transformation can be prefixed to existing transformations for the original schema in order to derive new transformations which apply to the new schema.

- *Transformations may be semi-automatically derived.* There is an obvious NP-hard algorithm which can generate candidate composite transformations from the set of primitive transformations, and then verify them against the actual database. A practical implementation of this approach could use heuristics such as the similarity of the names of schema items, or the frequency with which a given transformation has been applied in the past, together with user guidance, in order to limit the search space for new transformations.

In addition to developing a tool supporting the above features, for future work we wish to investigate further the applicability of our formalism to the wide range of schema integration methodologies that have been proposed. In particular, we wish to define the notion of a 'normal form' for an ER schema towards which transformation rules should be directed, and to investigate sets of complete and convergent transformation rules.

REFERENCES

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

[2] P. Atzeni, G. Ausiello, C. Batini, and M. Noscarini. Inclusion and equivalence between relational database schemata. *Theoretical Computer Science*, **19**:267–285 (1982).

[3] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, **10**(6):650–664 (1984).

[4] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, **18**(4):323–364 (1986).

[5] J. Biskup and B. Convent. A formal view integration method. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 398–407, Washington (1986).

[6] J. Biskup and U. Räsch. The equivalence problem for relational database schemes. In *Proceedings of the 1st Symposium on Mathematical Fundamentals of Database Systems*, volume 305 of *LNCS*, pp. 42–70, Springer-Verlag (1988).

[7] C. Francalanci and B. Pernici. View integration: A survey of current developments. Technical Report 93-053, Dipartimento di Elettronica e Informazione, P.zza Leonardo da Vinci 32, 20133 Milano, Italy (1993).

[8] S.J. Hegner. Unique complements and decompositions of database schemata. *Journal of Computer and System Sciences*, **48**(1):9–57 (1994).

[9] R. Hull. Relative information capacity of simple relational database scmemata. *SIAM Journal on Computing*, **15**(3):856–886 (1986).

[10] P. Johannesson. *Schema Integration, Schema Translation, and Interoperability in Federated Information Systems.* PhD thesis, DSV, Stockholm University, ISBN 91-7153-101-7, Rep. No. 93-010-DSV (1993).

[11] V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *VLDB Journal*, **5**(4):276–304 (1996).

[12] J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, **15**(4):449–463 (1989).

[13] P. McBrien and A. Poulovassilis. A general formal framework for schema transformation. Technical Report 98-05, King's College London, ftp://ftp.dcs.kcl.ac.uk/pub/tech-reports/tr98-05.ps.gz, To appear in Data & Knowledge Engineering (1998).

[14] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pp. 120–133, Trinity College, Dublin, Ireland (1993).

[15] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, **19**(1):3–31 (1994).

[16] S. Navathe and A. Savasere. A practical schema integration facility using an object-oriented approach. In A.Elmagarmid and O.Bukhres, editors, *Multidatabase Systems*. Prentice Hall (1996).

[17] J. Rissanen. Independent components of relations. *ACM Transactions on Database Systems*, **2**(4):317–325 (1977).

[18] A. Rosenthal and D. Reiner. Tools and transformations — rigorous and otherwise — for practical database design. *ACM Transactions on Database Systems*, **19**(2):167–211 (1994).

[19] A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, **22**(3):183–236 (1990).

[20] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. Technical report, Ecole Polytechnique Federale de Lausanne (1990).

[21] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogenous schemas. *The VLDB Journal*, **1**(1):81–126 (1992).

APPENDIX A: SEMANTICS OF THE PRIMITIVE TRANSFORMATIONS

Definition 11 below gives the semantics of each of the primitive transformations of Definition 6 by defining the changes it makes to the input model $\langle S, I, Ext_{S,I}\rangle$ to yield the output model $\langle S', I, Ext_{S',I}\rangle$. In this definition, it is assumed that $S = \langle Ents, Atts, Incs, Assocs\rangle$ and $S' = \langle Ents', Atts', Incs', Assocs'\rangle$. Also $Vars(q)$ denotes the set of variables in a query q.

Definition 11

1. $rename_X\ \langle fromName, toName\rangle\ \langle S, I, Ext_{S,I}\rangle\ =\ \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} S' &= S[fromName/toName] \\ Ext_{S',I}\langle o[fromName/toName]\rangle &= Ext_{S,I}(o) \end{aligned}$$

   provided that $toName$ is not the name of an entity, attribute or relationship in S.

2. $expand\ \langle n_0, n_1, n_2, c_1, c_2\rangle\ \langle S, I, Ext_{S,I}\rangle\ =\ \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} Assocs' &= (Assocs - OldAssoc) \cup \{\langle n_0, n_1, n_2, c_1, c_2\rangle\} \\ &\quad \text{where } OldAssoc = \{\langle n_0, n_1, n_2, oc_1, oc_2\rangle \mid \langle n_0, n_1, n_2, oc_1, oc_2\rangle \in Assocs\} \end{aligned}$$

   provided $OldAssoc \neq \{\}$, $oc_1 \subseteq c_1$ and $oc_2 \subseteq c_2$.

3. $contract\ \langle n_0, n_1, n_2, c_1, c_2\rangle\ \langle S, I, Ext_{S,I}\rangle\ =\ \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} Assocs' &= (Assocs - OldAssoc) \cup \{\langle n_0, n_1, n_2, c_1, c_2\rangle\} \\ &\quad \text{where } OldAssoc = \{\langle n_0, n_1, n_2, oc_1, oc_2\rangle \mid \langle n_0, n_1, n_2, oc_1, oc_2\rangle \in Assocs\} \end{aligned}$$

   provided $OldAssoc \neq \{\}$, $c_1 \subseteq oc_1, c_2 \subseteq oc_2$ and $Ext_{S,I}(\langle n_0, n_1, n_2\rangle)$ satisfies c_1 and c_2.

4. $add_E\ \langle e,q\rangle\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Ents' & = & Ents \cup \{e\} \\ Ext_{S',I}(e) & = & q[v_1/Ext_{S,I}(v_1),\ldots,v_n/Ext_{S,I}(v_n)] \\ & & \text{where } Vars(q)=\{v_1,\ldots,v_n\} \end{array}$$

provided that $e \notin Ents$ and $\{v_1,\ldots,v_n\} \subseteq Ents \cup Atts \cup Schemes$.

5. $del_E\ e\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Ents' & = & Ents - \{e\} \\ Ext_{S',I}(e) & = & \perp \end{array}$$

provided that e has no attributes and participates in no relationships and that property (ii) of Definition 2 is not violated by setting $Ext_{S',I}$ to be undefined for e.

6. $add_R\ \langle r,e_1,e_2,q\rangle\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Assocs' & = & Assocs \cup \{\langle r,e_1,e_2,c_1,c_2\rangle\} \\ Ext_{S',I}(\langle r,n_1,n_2\rangle) & = & q[v_1/Ext_{S,I}(v_1),\ldots,v_n/Ext_{S,I}(v_n)] \\ & & \text{where } Vars(q)=\{v_1,\ldots,v_n\} \end{array}$$

provided that $e_1,e_2 \in Ents$, $\langle r,e_1,e_2\rangle \notin Schemes$, $Vars(q) \subseteq Ents \cup Atts \cup Schemes$, and $q[v_1/Ext_{S,I}(v_1),\ldots,v_n/Ext_{S,I}(v_n)]$ satisfies the appropriate domain and cardinality constraints.

7. $del_R\ \langle r,e_1,e_2\rangle\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Assocs' & = & Assocs - OldAssoc \\ & & \text{where } OldAssoc = \{\langle r,e_1,e_2,c_1,c_2\rangle \mid \langle r,e_1,e_2,c_1,c_2\rangle \in Assocs\}) \\ Ext_{S',I}(\langle r,e_1,e_2\rangle) & = & \perp \end{array}$$

provided that $OldAssoc \neq \{\}$ and that property (ii) of Definition 2 is not violated by setting $Ext_{S',I}$ to be undefined for $\langle r,e_1,e_2\rangle$.

8. $add_A\ \langle e,a,c_1,c_2,q_{att},q_{assoc}\rangle\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Atts' & = & Atts \cup \{a\} \\ Assocs' & = & Assocs \cup \{\langle Null,e,a,c_1,c_2\rangle\} \\ Ext_{S',I}(a) & = & q_{att}[v_1/Ext_{S,I}(v_1),\ldots,v_n/Ext_{S,I}(v_n)] \\ & & \text{where } Vars(q_{att})=\{v_1,\ldots,v_n\} \\ Ext_{S',I}(\langle Null,e,a\rangle) & = & q_{assoc}[w_1/Ext_{S,I}(w_1),\ldots,w_m/Ext_{S,I}(w_m)] \\ & & \text{where } Vars(q_{assoc})=\{w_1,\ldots,w_m\} \end{array}$$

provided that $e \in Ents$, $\langle Null,e,a\rangle \notin Schemes$, $a \in Atts \Rightarrow Ext_{S',I}(a) = Ext_{S,I}(a)$, $Vars(q_{att}) \subseteq Ents \cup Atts \cup Schemes$, $Vars(q_{assoc}) \subseteq Ents \cup Atts \cup Schemes$, and $q_{assoc}[w_1/Ext_{S,I}(w_1),\ldots,w_m/Ext_{S,I}(w_m)]$ satisfies the appropriate domain and cardinality constraints.

9. $del_A\ \langle e,a\rangle\ \langle S,I,Ext_{S,I}\rangle\ =\ \langle S',I,Ext_{S',I}\rangle$ such that

$$\begin{array}{rcl} Atts' & = & Atts - \{a\}, \quad \text{if } \nexists\langle Null,e',a\rangle.\langle Null,e',a\rangle \in Schemes \wedge e' \neq e \\ & = & Atts, \quad \text{otherwise} \\ Assocs' & = & Assocs - OldAssoc \\ & & \text{where} \end{array}$$

$$\begin{aligned}
& \qquad OldAssoc = \{\langle Null, e, a, c_1, c_2\rangle | \langle Null, e, a, c_1, c_2\rangle \in Assocs\}) \\
Ext_{S',I}(\langle Null, e, a\rangle) &= \bot \\
Ext_{S',I}(a) &= \bot, \text{ if } \nexists \langle Null, e', a\rangle . \langle Null, e', a\rangle \in Schemes \wedge e' \neq e \\
&= Ext_{S,I}(a), \text{ otherwise}
\end{aligned}$$

provided that $OldAssoc \neq \{\}$ and that property (ii) of Definition 2 is not violated by setting $Ext_{S',I}$ to be undefined for $\langle Null, e, a\rangle$ and a.

10. $add_I \ \langle e_1, e_2\rangle \ \langle S, I, Ext_{S,I}\rangle \ = \ \langle S', I, Ext_{S',I}\rangle$ such that

$$Incs' \ = \ Incs \cup \{\langle e_1, e_2\rangle\}$$

provided $e_1, e_2 \in Ents$, $\langle e_1, e_2\rangle \notin transitiveClosure(Incs')$ and $Ext_{S,I}(e_1) \subseteq Ext_{S,I}(e_2)$.

11. $del_I \ \langle e_1, e_2\rangle \ \langle S, I, Ext_{S,I}\rangle \ = \ \langle S', I, Ext_{S',I}\rangle$ such that

$$Incs' \ = \ Incs - \{\langle e_1, e_2\rangle\}$$

provided $\langle e_1, e_2\rangle \in Incs$.

APPENDIX B: PROOF OF PROPOSITION 1

The proposition is proved by giving the reverse transformation for each of the primitive transformations of Definition 11:

1. $rename_X \ \langle fromName, toName\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $rename_X \ \langle toName, fromName\rangle \ \langle S', I, Ext_{S',I}\rangle$.

2. $expand \ \langle n_0, n_1, n_2, c_1, c_2\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $contract \ \langle n_0, n_1, n_2, oc_1, oc_2\rangle \ \langle S', I, Ext_{S',I}\rangle$, where $\langle n_0, n_1, n_2, oc_1, oc_2\rangle \in Assocs$.

3. $contract \ \langle n_0, n_1, n_2, c_1, c_2\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $expand \ \langle n_0, n_1, n_2, oc_1, oc_2\rangle \ \langle S', I, Ext_{S',I}\rangle$, where $\langle n_0, n_1, n_2, oc_1, oc_2\rangle \in Assocs$.

4. $add_E \ \langle e, q\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $del_E \ e \ \langle S', I, Ext_{S',I}\rangle$.

5. $del_E \ e \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $add_E \ \langle e, q\rangle \ \langle S', I, Ext_{S',I}\rangle$, where q is a query such that:

$$\begin{aligned}
Vars(q) &= \{v_1, \ldots, v_n\} \\
\{v_1, \ldots, v_n\} &\subseteq (Ents \cup Atts \cup Schemes) - \{e\} \\
Ext_{S,I}(e) &= q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]
\end{aligned}$$

Now we know that since the deletion transformation was successful, each set of I is derivable from the sets of $Range(Ext_{S',I})$. Thus, $Ext_{S,I}(e)$ is also derivable from the sets of $Range(Ext_{S',I})$ and so there exists a query such as q above.

6. $add_R \ \langle r, e_1, e_2, c_1, c_2, q\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $del_R \ \langle r, e_1, e_2\rangle \ \langle S', I, Ext_{S',I}\rangle$.

7. $del_R \ \langle r, e_1, e_2\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $add_R \ \langle r, e_1, e_2, c_1, c_2, q\rangle \ \langle S', I, Ext_{S',I}\rangle$, where $\langle c_1, c_2\rangle$ is the cardinality of $\langle r, e_1, e_2\rangle$ and q is a query such that:

$$\begin{aligned}
(Vars(q) = \{v_1, \ldots, v_n\}) &\subseteq (Ents \cup Atts \cup Schemes) - \{\langle r, e_1, e_2\rangle\} \\
Ext_{S,I}(\langle r, e_1, e_2\rangle) &= q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]
\end{aligned}$$

By the same argument as in item 5 above, we know that such a query q exists.

8. $add_A \ \langle e, a, c_1, c_2, q_{att}, q_{assoc}\rangle \ \langle S, I, Ext_{S,I}\rangle$ is reversed by $del_A \ \langle e, a\rangle \ \langle S', I, Ext_{S',I}\rangle$.

9. $del_A\ \langle e,a\rangle\ \langle S,I,Ext_{S,I}\rangle$ is reversed by $add_A\ \langle e,a,c_1,c_2,q_{att},q_{assoc}\rangle\ \langle S',I,Ext_{S',I}\rangle$ where $\langle c_1,c_2\rangle$ is the cardinality of $\langle Null,e,a\rangle$, and q_{att} and q_{assoc} are queries such that:

$$\begin{array}{rcl}
(Vars(q_{att}) = \{v_1,\ldots,v_n\}) & \subseteq & (Ents \cup Atts \cup Schemes) - \{a, \langle Null,e,a\rangle\}, \\
& & \quad \text{if } \nexists \langle Null,e',a\rangle . \langle Null,e',a\rangle \in Schemes \wedge e' \neq e \\
& \subseteq & (Ents \cup Atts \cup Schemes) - \{\langle Null,e,a\rangle\}, \quad \text{otherwise} \\
(Vars(q_{assoc}) = \{w_1,\ldots,w_m\}) & \subseteq & (Ents \cup Atts \cup Schemes) - \{a, \langle Null,e,a\rangle\}, \\
& & \quad \text{if } \nexists \langle Null,e',a\rangle . \langle Null,e',a\rangle \in Schemes \wedge e' \neq e \\
& \subseteq & (Ents \cup Atts \cup Schemes) - \{\langle Null,e,a\rangle\}, \quad \text{otherwise} \\
Ext_{S,I}(a) & = & q_{att}[v_1/Ext_{S,I}(v_1),\ldots,v_n/Ext_{S,I}(v_n)] \\
Ext_{S,I}(\langle Null,e,a\rangle) & = & q_{assoc}[w_1/Ext_{S,I}(w_1),\ldots,w_m/Ext_{S,I}(w_m)]
\end{array}$$

By the same argument as in items 5 and 7 above, we know that such queries q_{att} and q_{assoc} exist.

10. $add_I\ \langle e_1,e_2\rangle\ \langle S,I,Ext_{S,I}\rangle$ is reversed by $del_I\ \langle e_1,e_2\rangle\ \langle S',I,Ext_{S',I}\rangle$.

11. $del_I\ \langle e_1,e_2\rangle\ \langle S,I,Ext_{S,I}\rangle$ is reversed by $add_I\ \langle e_1,e_2\rangle\ \langle S',I,Ext_{S',I}\rangle$.