A Formal Framework for ER Schema Transformation

Peter M^{c}Brien and Alexandra Poulovassilis

Department of Computer Science, King's College London,
Strand, London WC2R 2LS.
alex,pjm@dcs.kcl.ac.uk

Abstract. Several methodologies for semantic schema integration have been proposed in the literature, often using some variant of the ER model as the common data model. As part of these methodologies, various transformations have been defined that map between ER schemas which are in some sense equivalent. This paper gives a unifying formalisation of the ER schema transformation process and shows how some common schema transformations can be expressed within this single framework. Our formalism clearly identifies which transformations apply for any instance of the schema and which only for certain instances.

1 Introduction

When data is to be shared or exchanged between heterogeneous databases, it is necessary to build a single integrated schema expressed using a common data model (CDM) [13]. Conflicts may exist between the export schemas of the component databases, which must be removed by performing transformations on the schemas to produce equivalent schemas.

In this paper we examine the behaviour of the schema transformations commonly found in the literature within a new formal framework, and distinguish in a precise manner between schema transformations which are dependent on knowledge about the instances associated with the schema, and those which are not. This distinction has the benefit of precisely defining what assumptions are made when a database object is transformed or is considered to have the same "real world state" [8] as some other object.

Previous work on schema transformation has either been to some extent informal [2, 7, 6], has formalised only transformations that are independent of database content [10, 11], or is limited to certain types of transformation only [3, 8, 14, 5]. The latter cases assume that specific types of dependency constraints are employed to limit the instances of schemas in order that the schemas can be regarded as equivalent. In contrast, our approach allows arbitrary constraints on instances to be specified as part of the transformation rules. A similar approach has recently been adopted in [6] where the notion of a database "context" constrains instances so that schemas can be considered equivalent.

In common with much previous work on semantic schema integration [1, 2, 4], we use a variant of the ER model as the CDM, namely a binary ER model with subtypes. This model supports entity types with attributes, subtype relationships between entity types, and binary relationships (without attributes) between entity types. Subtype relationships give sufficient modelling expressiveness for representing object-oriented schemas. By omitting generalisation hierarchies, aggregations and n-ary relationships for n > 2, some alternate representations of the universe of discourse are avoided. There is no loss of modelling expressiveness since there are obvious transformations that can be applied from these more complex structures into our CDM (see, for example, [9]).

Similarly to [1], in our CDM each relationship, r, between two entity types, e_1 and e_2, has associated with it a pair of cardinality constraints, $l_1 : u_1$ and $l_2 : u_2$, where $l_1 : u_1$ ($l_2 : u_2$) indicates the lower and upper cardinalities of the participation in r by each instance of e_1 (e_2). Each association between an entity type and an attribute similarly has a pair of cardinality constraints associated with it. If not explicitly stated, a cardinality constraint of $1{:}1$ should be assumed.

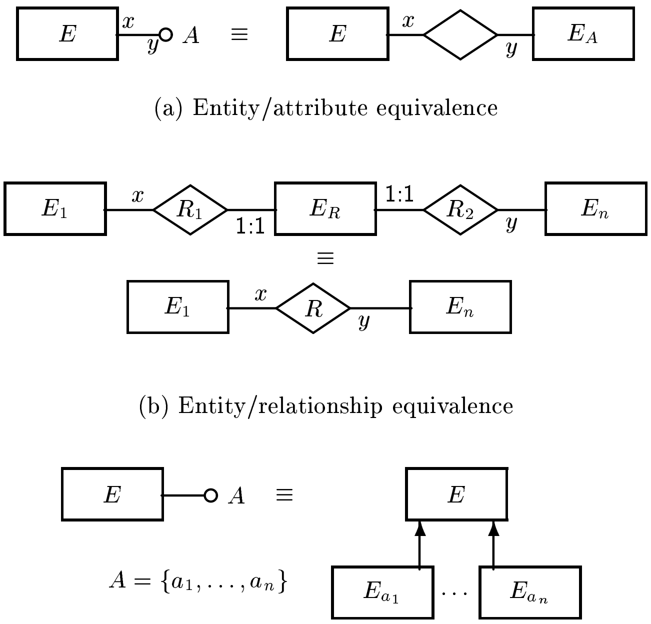
The main tasks of database schema integration are **pre-integration**, **schema conforming**, **schema merging** and **schema restructuring** [2]. The last three of these tasks involve a process of **schema transformation**, and Figures 1–3 illustrate some of the common transformations used. In practice, schema conforming transformations are applied bi-directionally and schema merging and restructuring ones uni-directionally. For each of these transformations, the original and resulting schema obey one or more alternative notions of schema **equivalence** [12, 1, 8]. This paper presents a unifying formalism for the schema transformation process, including all of the transformations of Figures 1–3.

In Section 2 we define the notions of ER schemas, instances and models. In Section 3 we define the notion of schema equivalence which provides the semantic foundation of schema transformation in our approach. In Section 4 we define a set of primitive transformations and explore their properties with respect to schema equivalence. We then extend these transformations into "knowledge-based" versions, which allow conditions on instances to be expressed. Section 5 demonstrates the expressiveness of our primitive transformations by showing how they can be used to define all the transformations of Figures 1–3. In Section 6 we give our concluding remarks.

2 ER Schemas, Instances and Models

Our definitions of schemas, instances and models (Definitions 1-3 below), assume the availability of two disjoint sets: $Vals$ (values) and $Names$ (names of entity types, attributes and relationships). The distinguished constant $Null$ is a member of $Names$. $Seq(Vals)$ is the set of sequences of values of finite length. P is the powerset operator. $Cards$ is the set of cardinality constraints, a cardinality constraint being a pair $l{:}u$, where l is a natural number and u is either a natural number or N (denoting no upper limit). For any function f, $Range(f)$ denotes the range of f.

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



(a) Entity/attribute equivalence

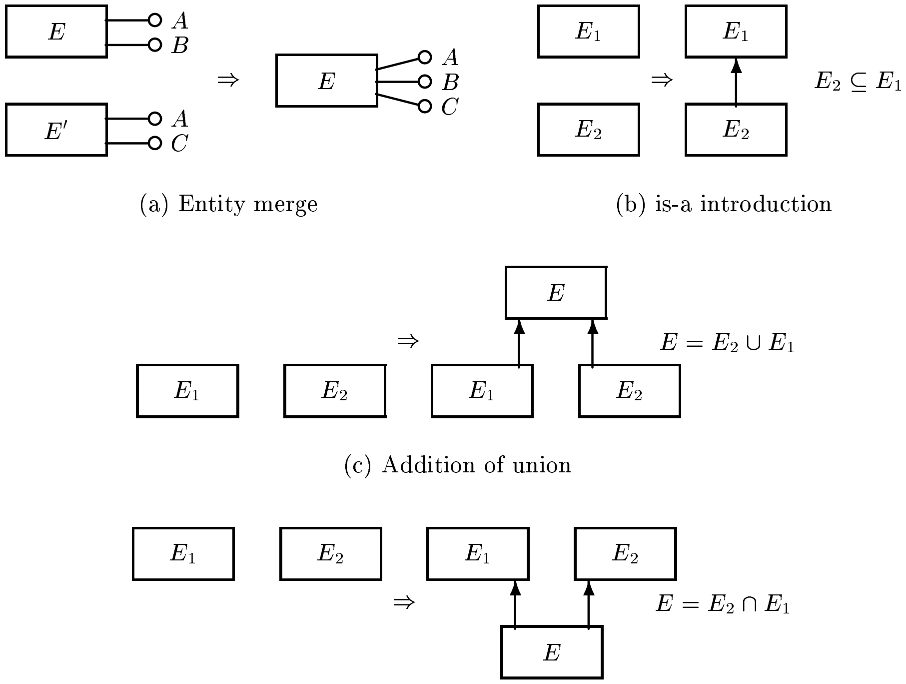
(b) Entity/relationship equivalence

(c) Attribute/subtype equivalence

Fig. 1. Schema Conforming Transformations

- $Ents \subseteq Names$ 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**.
  For each relationship between two entity types $e_1, e_2 \in Ents$, there is a tuple $\langle rel\_name, e_1, e_2, c_1, c_2 \rangle \in Assocs$, where c_1 indicates the lower and upper cardinalities of instances of e_2 for each instance of e_1, and 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.
  Similarly, for each attribute a associated with an entity type e there is a tuple $\langle Null, e, a, c_1, c_2 \rangle \in Assocs$, where 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.

Definition 2 below defines an instance, I, of a schema, S, to be a set of sets. The extent of each item in S (*i.e.* each entity, attribute and association) is derived by means of an expression over the sets of I expressed in some mapping language



(a) Entity merge

(b) is-a introduction

(c) Addition of union

(d) Addition of intersection

Fig. 2. Schema Merging Transformations

L (point (i) below). In order to support updates to I, this mapping should be reversible (point (ii)). The instance I respects the subtype relationships in S (point (iii)) and the domain and cardinality constraints (point (iv)).

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

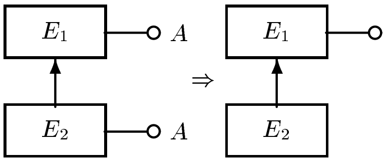
$$Schemes = \{\langle n_0, n_1, n_2 \rangle \mid \langle n_0, n_1, n_2, c_1, c_2 \rangle \in Assocs\}$$

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

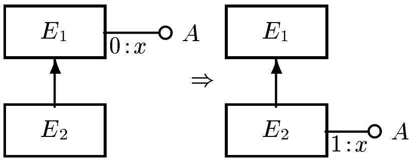
$$Ext_{S,I} : Ents \cup Atts \cup Schemes \to P(Seq(Vals))$$

where:

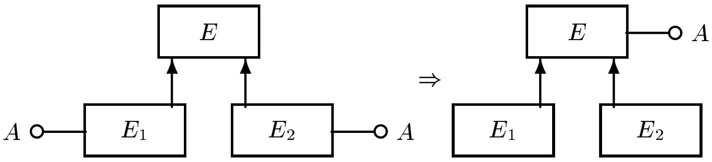
(i) each set in $Range(Ext_{S,I})$ is derivable by means of an expression in L over the sets of I;
(ii) conversely, each set in I is derivable by means of an expression in L over the sets of $Range(Ext_{S,I})$;
(iii) for any $\langle e_1, e_2 \rangle \in Incs$, $Ext_{S,I}(e_1) \subseteq Ext_{S,I}(e_2)$;



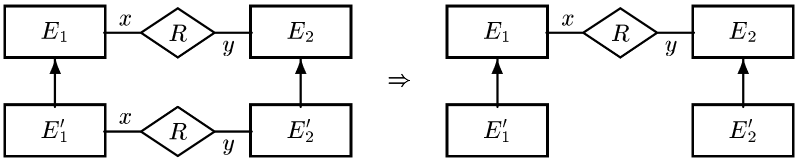
(a) Redundant attribute removal



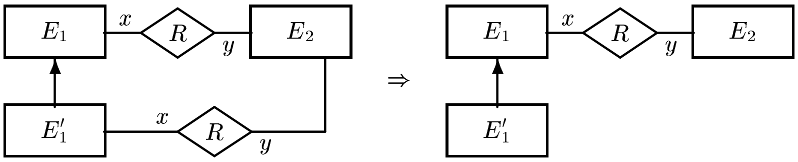
(b) Optional attribute removal



(c) Generalisation of attributes



(d) Redundant relationship removal (1)



(e) Redundant relationship removal (2)

Fig. 3. Schema Restructuring Transformations

(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 of $\langle n_0, n_1, n_2\rangle$ is $\langle l_1 : u_1, l_2 : u_2\rangle$ then
$\forall s_1 \in Ext_{S,I}(n_1).l_1 \leq \mid \{n_2(s) | s \in Ext_{S,I}(\langle n_0, n_1, n_2\rangle) \wedge n_1(s) = s_1\} \mid \leq u_1$
and

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

Such a function $Ext_{S,I}$ is termed an **extension mapping** from S to I.

Definition 3. A **model** is a triple $\langle S, I, Ext_{S,I}\rangle$ where S is a schema, I is an instance of S and $Ext_{S,I}$ an extension mapping from S to I.

3 Schema Equivalence

Definition 4. $Inst(S)$ denotes 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 **equivalent** if $Inst(S') = Inst(S)$.

For example, the top half of Figure 4 shows a schema S consisting of an entity 'person' with an attribute 'dept', a database instance I consisting of three sets $\{john, jane, mary\}$, $\{compsci, maths\}$ and $\{\langle john, compsci\rangle, \langle jane, compsci\rangle, \langle jane, maths\rangle, \langle mary, maths\rangle\}$, and the function $Ext_{S,I}$ defined as follows:

$$\begin{aligned}
Ext_{S,I}(\text{person}) &= \{john, jane, mary\} \\
Ext_{S,I}(\text{dept}) &= \{compsci, maths\} \\
Ext_{S,I}(\langle Null, \text{person}, \text{dept}\rangle) &= \{\langle john, compsci\rangle, \langle jane, compsci\rangle, \\
&\quad\ \langle jane, maths\rangle, \langle mary, maths\rangle\}
\end{aligned}$$

The bottom half of Figure 4 shows another schema S' consisting of two entities 'person' and 'dept' and a relationship 'works_in' between them. S' subsumes S in the sense that any instance of S is also an instance of S'. In particular, we can define $Ext_{S',I}$ in terms of $Ext_{S,I}$ as follows:

$$\begin{aligned}
Ext_{S',I}(\text{person}) &= Ext_{S,I}(\text{person}) \\
Ext_{S',I}(\text{dept}) &= Ext_{S,I}(\text{dept}) \\
Ext_{S',I}(\langle \text{works_in}, \text{person}, \text{dept}\rangle) &= Ext_{S,I}(\langle Null, \text{person}, \text{dept}\rangle)
\end{aligned}$$

By a similar argument, it is easy to see that S subsumes S', and so S and S' are equivalent. This is an example of the entity/attribute equivalence of Figure 1(a).

We can generalise the definition of equivalence to incorporate a condition on the instances of one or both schemas:

Definition 5. $Inst(S, f)$ denotes the set of instances of a schema S that satisfy a given condition f. A schema S **conditionally subsumes (c-subsumes)** a schema S' w.r.t. f if $Inst(S', f) \subseteq Inst(S, f)$. Two schemas S and S' are **conditionally equivalent (c-equivalent) w.r.t** f if $Inst(S, f) = Inst(S', f)$.

For example, in Figure 5 the schema S and the instance I are as in Figure 4. The schema S' now consists of three entities 'person', 'mathematician' and 'computer scientist', with the last two being subtypes of the first. I can be shown to be an instance of S' only if the domain of the 'dept' attribute consists of two values. In this case this is indeed so, the two values being *compsci* and *maths* and we can define $Ext_{S',I}$ in terms of $Ext_{S,I}$ as follows:

$$\begin{aligned} Ext_{S',I}(\text{person}) &= Ext_{S,I}(\text{person}) \\ Ext_{S',I}(\text{mathematician}) &= \{x \mid \langle x, maths\rangle \in Ext_{S,I}(\langle Null, \text{person}, \text{dept}\rangle)\} \\ Ext_{S',I}(\text{computer scientist}) &= \{x \mid \langle x, compsci\rangle \in Ext_{S,I}(\langle Null, \text{person}, \text{dept}\rangle)\} \end{aligned}$$

Conversely, we can define $Ext_{S,I}$ in terms of $Ext_{S',I}$ as follows:

$$\begin{aligned} Ext_{S,I}(\text{person}) &= Ext_{S',I}(\text{person}) \\ Ext_{S,I}(\text{dept}) &= \{maths, compsci\} \\ Ext_{S,I}(\langle Null, \text{person}, \text{dept}\rangle) &= \{\langle x, maths\rangle \mid x \in Ext_{S',I}(\text{mathematician})\} \ \cup \\ &\quad\ \{\langle x, compsci\rangle \mid x \in Ext_{S',I}(\text{computer scientist})\} \end{aligned}$$

Thus S and S' are c-equivalent with respect to the condition that $\mid Ext_{S,I}(\text{dept}) \mid = 2$. This is an example of the attribute/subtype equivalence of Figure 1(c).

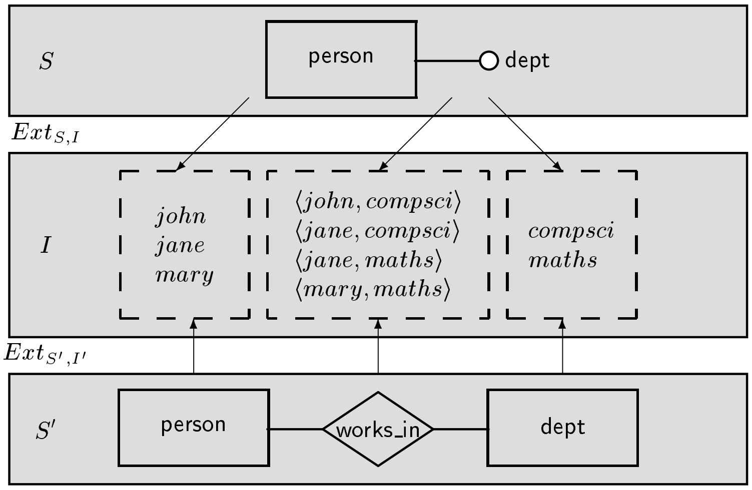


Fig. 4. Two equivalent schemas

4 Transformation of ER Models

We now propose a set of primitive transformations on ER models and explore their properties with respect to schema equivalence. Each primitive transformation takes an argument and a model and returns a new model. The instance component of the input model is left unchanged, only the schema component

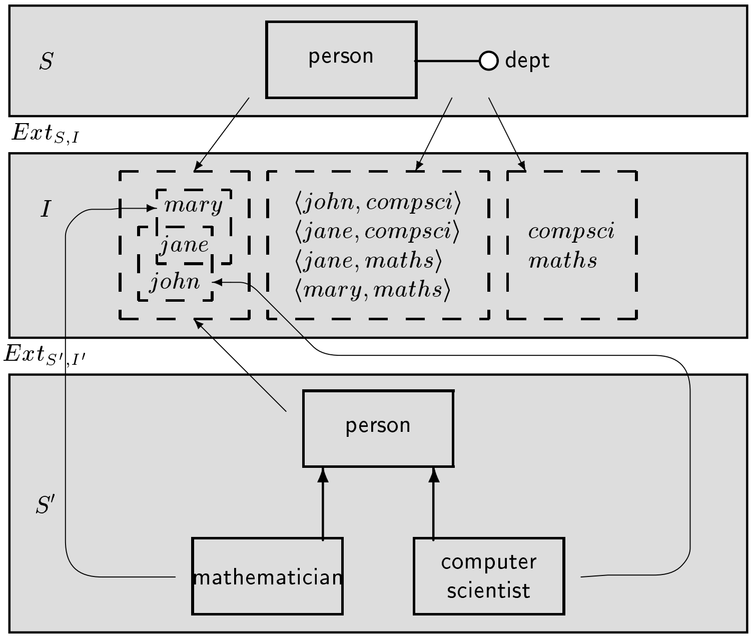


Fig. 5. Two c-equivalent schemas

and the extension mapping are changed. Each primitive transformation has one or more provisos associated with it. If these are not met then the transformation returns an "undefined" value, denoted by $\emptyset$. The result of applying a primitive transformation to $\emptyset$ is $\emptyset$.

Definition 6 below lists the primitive transformations, giving their name and the type of their first argument and explaining their effect. Their full formal definition can be found in [9]. In Definition 6, *Queries* is the set of queries expressible in the mapping language L. Such queries will be applied to items of the input schema (*i.e.* to members of $Ents \cup Atts \cup Schemes$) and will return a set which is the extent of the new item.

Definition 6. The following are the primitive transformations.

1. $rename_X(Names \times Names)$ renames a schema item, where 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 have been replaced by $toName$. **Proviso:** $toName$ is not already the name of an entity, attribute or relationship in the schema.
2. $expand(Names \times Names \times Names \times Cards \times Cards)$ expands a constraint. It takes a tuple $\langle n_0, n_1, n_2, c_1, c_2 \rangle$ and a model, and replaces the old cardinality constraint on this association by the new, relaxed, constraint $c_1 : c_2$.

Provisos: $\langle n_0, n_1, n_2, oc_1, oc_2 \rangle$ is an existing association for some oc_1 and oc_2, $oc_1 \subseteq c_1$, and $oc_2 \subseteq c_2$.

3. $contract(Names \times Names \times Names \times Cards \times Cards)$ contracts a constraint. It similarly takes a tuple $\langle n_0, n_1, n_2, c_1, c_2 \rangle$ and a model, and replaces the old cardinality constraint on this association by the new, stricter, constraint $c_1 : c_2$. **Provisos:** $\langle n_0, n_1, n_2, oc_1, oc_2 \rangle$ is an existing association for some oc_1 and oc_2, $c_1 \subseteq oc_1$, $c_2 \subseteq oc_2$, and $Ext_{S,I}(\langle n_0, n_1, n_2 \rangle)$ satisfies c_1 and c_2.
4. $add_E(Names \times Queries)$ adds a new entity.
   It takes a pair $\langle e, q \rangle$ and a model, and adds e to the entity set of the schema. e's extent is given by the value of the query q. **Proviso:** e is a new entity.
5. $del_E(Names)$ deletes an entity.
   It takes an entity e and a model, and removes e from the schema. **Provisos:** e has no attributes and participates in no relationships, and property (ii) of Definition 2 is not violated by setting $Ext_{S',I}(e)$ to be undefined.
6. $add_R(Names \times Names \times Names \times Cards \times Cards \times Queries)$ adds a new relationship.
   It takes a tuple $\langle r, e_1, e_2, c_1, c_2, q \rangle$ and a model, and adds $\langle r, e_1, e_2, c_1, c_2 \rangle$ to the set of associations of the schema. The extent of this relationship is given by the value of the query q. **Provisos:** entities e_1 and e_2 exist, r is a new relationship between them and its extent satisfies the appropriate domain and cardinality constraints.
7. $del_R(Names \times Names \times Names)$ removes a relationship.
   It takes a tuple $\langle r, e_1, e_2 \rangle$ and a model, and removes this relationship from the schema. **Provisos:** this relationship exists, and property (ii) of Definition 2 is not violated by setting $Ext_{S',I}(\langle r, e_1, e_2 \rangle)$ to be undefined.
8. $add_A(Names \times Names \times Cards \times Cards \times Queries \times Queries)$ adds an attribute to an entity type. The attribute may either already exist in the schema or may be a new attribute.
   It takes a tuple $\langle e, a, c_1, c_2, q_{att}, q_{assoc} \rangle$ and a model, and adds $\langle Null, e, a, c_1, c_2 \rangle$ to the set of associations of the schema. The queries q_{att} and q_{assoc} respectively give the extents of the attribute and the association. **Provisos:** entity e exists and does not already have an attribute a, and the extent of the association satisfies the appropriate domain and cardinality constraints.
9. $del_A(Names \times Names)$ removes an attribute from an entity type.
   It takes a pair $\langle e, a \rangle$ and a model, and removes this association from the schema. **Provisos:** this association exists, and property (ii) of Definition 2 is not violated by setting $Ext_{S',I}$ to be undefined for $\langle Null, e, a \rangle$ and for a.
10. $add_I(Names \times Names)$ adds a subtype relationship.
    It takes a pair of entities $\langle e_1, e_2 \rangle$ and a model, and adds this subtype relationship to the schema. **Provisos:** entities e_1 and e_2 exist, $\langle e_1, e_2 \rangle \notin transitiveClosure(Incs')$ and $Ext_{S,I}(e_1) \subseteq Ext_{S,I}(e_2)$.
11. $del_I(Names \times Names)$ removes a subtype relationship.
    It takes a pair of entities $\langle e_1, e_2 \rangle$ and a model, and removes this subtype relationship from the schema. **Proviso:** this subtype relationship exists in the schema.

We note that the above primitive transformations are syntactically complete, in the sense that without their associated provisos they could be used to transform any ER schema into any other ER schema. With the addition of the provisos, the transformations become semantically sound *i.e.* they output a valid model.

We also note that 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 from applying the primitive transformation t to any model of the schema S.

Definition 7. A primitive transformation t is **schema-dependent (s-d) w.r.t. a schema** S if t does not return $\emptyset$ for any model of S, otherwise t is **instance-dependent (i-d) w.r.t.** S.

It is easy to see that if t is s-d w.r.t. S then $Schema(t, S)$ subsumes S. Thus, if a schema S can be transformed to a schema S' by means of a series of s-d primitive transformations, and vice versa, then S and S' are equivalent. Similarly, if t is i-d w.r.t. S with associated proviso f then $Schema(t, S)$ c-subsumes S w.r.t. f. Thus, if a schema S can be transformed to a schema S' by means of a series of s-d or i-d primitive transformations with an overall proviso f, and vice versa, then S and S' are c-equivalent w.r.t f.

For each of the primitive transformations of Definition 6 we now define a new transformation that takes as an extra argument a condition which must be satisfied in order for the transformation to be successful. We call such transformations **knowledge-based** (k-b) ones. We use the same name for both the 2-parameter and the 3-parameter versions of the primitive transformations since the number of arguments distinguishes which version is being used. Each 3-parameter version, *op*, is defined in terms of the 2-parameter one as follows:

$$op\ arg\ f\ m \quad = \quad \text{if } f(m) \text{ then } (op\ arg\ m) \text{ else } \emptyset$$

We note that, semantically, there is no difference between i-d and k-b transformations since both require instances to satisfy a condition.

5 Expressiveness of the primitive transformations

We now show how the transformations of Figures 1–3 can be expressed using sequences of our primitive transformations.

Entity/Attribute Equivalence. If a schema S contains an attribute a, an entity e, and an association $\langle Null, e, a, c_1, c_2\rangle$, then the following sequence of s-d primitive transformations converts 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 a schema S' contains an entity e_a whose only association is $\langle r, e, e_a, c_1, c_2\rangle$, then the following sequence of s-d primitive transformations converts 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, by the remarks in Section 4, S and S' are 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 $\langle r_2, e_R, e_2, 1:1, c_2\rangle$, then the following sequence of s-d primitive transformations transforms e_R into a new relationship r:

$$\mathit{add}_R\ \langle r, e_1, e_2, c_1, c_2, \{\langle e_1(s_1), e_R(s_1), e_2(s_2)\rangle \mid s_1 \in \langle r_1, e_1, e_R\rangle \wedge s_2 \in \langle r_2, e_R, e_2\rangle \\ \wedge e_R(s_1) = e_R(s_2)\}\rangle;\ \mathit{del}_R\ \langle r_1, e_1, e_R\rangle;\ \mathit{del}_R\ \langle r_2, e_R, e_2\rangle;\ \mathit{del}_E\ e_R$$

Conversely, if S' contains a relationship $\langle r, e_1, e_2, c_1, c_2\rangle$, then the following sequence of s-d primitive transformations transforms r into a new entity e_R and new relationships r_1, r_2:

$$\mathit{add}_E\ \langle e_R, \langle r, e_1, e_2\rangle\rangle;\ \mathit{add}_R\ \langle r_1, e_1, e_R, c_1, 1{:}1, \langle r, e_1, e_2\rangle\rangle; \\ \mathit{add}_R\ \langle r_2, e_R, e_2, 1{:}1, c_2, \langle r, e_1, e_2\rangle\rangle;\ \mathit{del}_R\ \langle r, e_1, e_2\rangle$$

Thus, S and S' are equivalent.

Attribute/subtype equivalence. If S contains an attribute a, an entity e, and an association between them $\langle \mathit{Null}, e, a, m_1:n_1, m_2:n_2\rangle$, then we can create n subtypes of e, and remove a, where the condition f is $(\mathit{Ext}_{S,I}(a) = \{a_1, \ldots, a_n\})$:

$$\mathit{add}_E\ \langle e_1, \{e(s) \mid s \in \langle \mathit{Null}, e, a\rangle \wedge a(s) = a_1\}\rangle;\ \ldots; \\ \mathit{add}_E\ \langle e_n, \{e(s) \mid s \in \langle \mathit{Null}, e, a\rangle \wedge a(s) = a_n\}\rangle; \\ \mathit{add}_I\ \langle e_1, e\rangle;\ \ldots;\ \mathit{add}_I\ \langle e_n, e\rangle;\ \mathit{del}_A\ \langle e, a\rangle\ f$$

Conversely, if S' contains entity types $e, e_1, \ldots, e_n$, where $e_1, \ldots, e_n$ are subtypes of e and participate in no associations, then we can replace $e_1, \ldots, e_n$ by a new attribute a of e:

$$\mathit{add}_A\ \langle e, a, m_1{:}n_1, m_2{:}n_2, \{a_1, \ldots, a_n\}, \cup_{i=1\ldots n}\{\langle x, a_i\rangle \mid x \in e_i\}\rangle\ f'; \\ \mathit{del}_I\ \langle e_n, e\rangle;\ \ldots;\ \mathit{del}_I\ \langle e_1, e\rangle;\ \mathit{del}_E\ e_n;\ \ldots;\ \mathit{del}_E\ e_1$$

where the condition f' is:

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

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

Entity merge. We can rename an entity e' to an existing entity e if they have the same extent, and their common associations also have the same extents:

$$\mathit{rename}_E\ \langle e', e\rangle\ (\mathit{Ext}_{S,I}(e) = \mathit{Ext}_{S,I}(e') \wedge \\ (\forall \langle n, e, \mathit{to}\rangle \in \mathit{Schemes}. \\ (\langle n, e', \mathit{to}\rangle \in \mathit{Schemes} \Rightarrow \mathit{Ext}_{S,I}(\langle n, e, \mathit{to}\rangle) = \mathit{Ext}_{S,I}(\langle n, e', \mathit{to}\rangle))\ \wedge \\ (\langle n, \mathit{to}, e'\rangle \in \mathit{Schemes} \Rightarrow \mathit{Ext}_{S,I}(\langle n, e, \mathit{to}\rangle) = \mathit{Ext}_{S,I}(\langle n, \mathit{to}, e'\rangle)))\ \wedge \\ (\forall \langle n, \mathit{to}, e\rangle \in \mathit{Schemes}.$$

$$(\langle n, e', to\rangle \in Schemes \Rightarrow Ext_{S,I}(\langle n, to, e\rangle) = Ext_{S,I}(\langle n, e', to\rangle)) \wedge$$
$$(\langle n, to, e'\rangle \in Schemes \Rightarrow Ext_{S,I}(\langle n, to, e\rangle) = Ext_{S,I}(\langle n, to, e'\rangle)))) \wedge$$
$$(\forall \langle n, e, e\rangle \in Schemes.$$
$$(\langle n, e', e'\rangle \in Schemes \Rightarrow Ext_{S,I}(\langle n, e, e\rangle) = Ext_{S,I}(\langle n, e', e'\rangle))))$$

This transformation is reversible by a sequence of s-d transformations which recreate the entity e', re-attach all its original attributes and relationships and detach the necessary attributes and relationships from e'. So the original and transformed schemas are c-equivalent.

Introduction of a union or intersection entity. If S contains two entities e_1, e_2, 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 we can apply to following sequence of primitive transformations to obtain S, provided $Ext_{S',I}(e) = Ext_{S',I}(e_1)$ $\cup Ext_{S',I}(e_2)$:

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

Thus S and S' are c-equivalent w.r.t the above proviso. Similar remarks apply to the introduction of an intersection entity $e_1 \cap e_2$ into S, and to its removal from S' provided $Ext_{S',I}(e) = Ext_{S',I}(e_1) \cap Ext_{S',I}(e_2)$.

The rest of the transformations we give here are reversible by s-d transformations and so give rise to c-equivalent schemas:

Subtype relationship introduction. If S contains entities e and e', then $add_I\ \langle e', e\rangle$ introduces a subtype relationship between them provided $Ext_{S,I}(e') \subseteq Ext_{S,I}(e)$.

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 $\langle 0{:}x, c\rangle$ can be moved from an entity e to a set of its subtypes $e_1, \ldots e_n$ if a is only ever associated with these subtypes and is a mandatory attribute for these types:

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

Generalisation of attributes. If entities $e_1, \ldots e_n$ have a common super-entity e and their extents cover the extent of e, then an attribute a common to $e_1, \ldots, e_n$ can 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 can 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 in Figure 3(e):

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

6 Concluding Remarks

In this paper we have proposed a formal framework for ER schema transformation. We have defined a set of primitive transformations and have demonstrated their expressiveness by showing how they can be used to define the schema transformations commonly found in the literature. The notion of schema equivalence which underpins our primitive transformations is based on formalising a database instance as a set of sets. We have distinguished between transformations which apply for any instance of a schema and those which only apply for certain instances. Our work is novel in that previous work on schema transformation has either been informal, or has formalised only transformations that are independent of the database instance, or is limited to specific types of transformation. A detailed theoretical treatment of our notion of schema equivalence can be found in [9], as well as a discussion of how our approach can be applied to the overall schema integration process.

Our work has practical application in the implementation of tools for aiding schema integration. Our primitive transformations can be used as a "programming language" for the specification of new schema transformations to suit particular environments. The distinction between s-d, and i-d and k-b transformations serves to identify which transformations need to be verified against the data, and possibly other knowledge about the database (*e.g.* semantic integrity constraints), to ensure that they can be applied. 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.

References

1. 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, November 1984.
2. C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
3. J. Biskup and B. Convent. A formal view integration method. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 398–407, Washington, 1986. ACM.
4. 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.
5. P. Johannesson. *Schema Integration, Schema Translation, and Interoperability in Federated Information Systems*. PhD thesis, DSV, Stockholm University, 1993. ISBN 91-7153-101-7, Rep. No. 93-010-DSV.
6. V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *VLDB Journal*, 5(4):276–304, 1996.
7. W. Kim, I. Choi, S. Gala, and M. Scheeval. On resolving schematic heterogeneity in multidatabase systems. In *Modern Database Systems*. ACM Press, 1995.
8. 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, April 1989.
9. P. McBrien and A. Poulovassilis. A formalisation of semantic schema integration. Technical Report 96-01, King's College London, ftp://ftp.dcs.kcl.ac.uk/pub/tech-reports/tr96-01.ps.gz, 1996.
10. 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*, pages 120–133, Trinity College, Dublin, Ireland, 1993.
11. 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.
12. J. Rissanen. Independent components of relations. *ACM Transactions on Database Systems*, 2(4):317–325, December 1977.
13. A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.
14. S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogenous schemas. *The VLDB Journal*, 1(1):81–126, 1992.

This article was processed using the LaTeX macro package with LLNCS style