From: Data & Knowledge Engineering
Volume 28, Pages 47--71,
Elsevier, 1998

A General Formal Framework for Schema Transformation

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

Abstract

Several methodologies for integrating database schemas have been proposed in the literature, using various common data models (CDMs). As part of these methodologies transformations have been defined that map between schemas which are in some sense equivalent. This paper describes a general framework for formally underpinning the schema transformation process. Our formalism clearly identifies which transformations apply for any instance of the schema and which only for certain instances. We illustrate the applicability of the framework by showing how to define a set of primitive transformations for an extended ER model and by defining some of the common schema transformations as sequences of these primitive transformations. The same approach could be used to formally define transformations on other CDMs.

Keywords: Schema integration. Schema transformation. Schema equivalence.

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) [15]. 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 schema transformation process within a new formal framework that distinguishes in a precise manner between schema transformations which are dependent on knowledge about the instances of the schema, and those which are not. This distinction has the advantage 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.

In [11] we assumed as the CDM a binary ER model with subtypes. We defined the notions of ER schemas and instances, and of equivalence of ER schemas. We defined a set of primitive transformations on ER schemas and explored their properties with respect to schema equivalence. We demonstrated the expressiveness of these primitive transformations by showing how they can be used to express many of the common schema equivalences found in the literature, thereby formally deriving precisely what, if any, knowledge about instances these equivalences are dependent upon.

This paper extends [11] in two ways. Firstly, recognising the fact that different methodologies might employ different CDMs, we take a step back and define a very general notion of a schema as a hypergraph. Schemas defined using a specific CDM can then be regarded as higher-level abstractions of the underlying hypergraph, together with additional constraints that must be satisfied by all instances of the schema. We develop the notions of instances of schemas and schema equivalence at this lower level of abstraction, and we define a set of primitive transformations on schemas. We then illustrate how a higher-level CDM and transformations on it can be defined in this framework by showing how to define the binary ER schemas and primitive transformations on them that we considered in [11].

The second extension of the paper is to further demonstrate the applicability of our framework by defining a much richer CDM, namely an ER model supporting n-ary relations, attributes on relations, complex attributes, and generalisation hierarchies. We define a set of primitive transformations for this model and show how they can be used to express many of the common schema equivalences regarding n-ary relations, attributes of relations, complex attributes and generalisation hierarchies found in the literature.

The structure of this paper is as follows. In Section 2 we define schemas, instances and models, and the notion of schema equivalence which provides the semantic foundation of our schema transformations. In Section 3 we define a set of primitive transformations and explore their properties with respect to schema equivalence. We next extend these transformations into "knowledge-based" versions, which allow conditions on instances to be expressed. We then extend the treatment to composite transformations comprising a sequence of primitive transformations. Section 4 demonstrates the applicability of this framework by first showing how to define the binary ER schemas and primitive transformations on them that we considered in [11], and then defining a much richer CDM and transformations thereon. Section 5 shows how many of the common schema equivalences on this richer CDM can be expressed in terms of these transformations, and formally derives precisely what knowledge about instances these equivalences are dependent upon. Section 6 briefly compares our approach with related work and Section 7 gives our concluding remarks.

2 The Formalism

2.1 Schemas, Instances and Models

Before proceeding to formally define these notions we require some auxiliary definitions. In particular, we assume the availability of two disjoint sets, $Vals$ (values) and $Names$ (the names of nodes and edges). The set $Schemes$ is defined recursively as follows:

- $Names \subseteq Schemes$
- $\langle n_0, n_1, \ldots, n_m \rangle \in Schemes$ if $m \geq 1$, $n_0 \in Names$, and $n_i \in Schemes$ for all $1 \leq i \leq m$.

The distinguished constant $Null$ is a valid name. For any set T, $Seq(T)$ denotes the set of finite sequences of members of T.

Definition 1 A **schema**, S, is a triple $\langle Nodes, Edges, Constraints \rangle$ where:

- $Nodes \subseteq Names$
- $Edges \subseteq Names \times Seq(Schemes)$ such that for any $\langle n_0, n_1, \ldots, n_m \rangle \in Edges$, $n_i \in Nodes \cup Edges$ for all $1 \leq i \leq m$.
- $Constraints$ is a set of boolean-valued expressions whose variables are members of $Nodes \cup Edges$.

Thus the first two components of a schema define a labelled, directed, nested hypergraph (nested in the sense that hyperedges can themselves participate in hyperedges). The third component of a schema states any extra constraints that all instances of the schema must satisfy.

We define an **instance** of a schema in Definition 2 below. An instance is not an absolute notion but is related to 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. From this, an extent for each scheme in the schema can be derived by means of an expression in the mapping language L over the sets of I (point (i) below). In order to support updates to the instance, this mapping should 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's nodes and edges (point (ii) below). The instance should satisfy the appropriate domain constraints (point (iii)) as well as any additional constraints in the schema (point (iv)):

Definition 2 Given a schema $S = \langle Nodes, Edges, Constraints \rangle$, an **instance of** S is a set $I \subseteq P(Seq(Vals))$ such that there exists a function

$$Ext_{S,I} : Nodes \cup Edges \rightarrow P(Seq(Vals))$$

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) each sequence $s \in Ext_{S,I}(\langle n_0, n_1, \ldots, n_m \rangle)$ contains m subsequences $s_1, \ldots, s_m$ where $s_i \in Ext_{S,I}(n_i)$ for all $1 \leq i \leq m$; we use $n_i(s)$ to denote the subsequence of s corresponding to the scheme n_i, for $1 \leq i \leq m$;

(iv) for every $c \in Constraints$, the expression $c[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$ evaluates to true, where $v_1, \ldots, v_n$ are the variables of c.

We call such a function $Ext_{S,I}$ 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 $Ext_{S,I}$ 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 Equivalence of 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 $Inst(S') = Inst(S)$.

Since it is defined in terms of instances of schemas, u-equivalence is not absolute but depends on the expressiveness of the mapping language, L. If we regard $Range(Ext_{S,I})$ as the extension of the schema S, u-equivalence implies that each extension of S can be derived from an extension of S' and vice versa. To see why this is so consider two u-equivalent schemas $S = \langle Nodes, Edges, Constraints \rangle$ and $S' = \langle Nodes', Edges', Constraints' \rangle$, and a pair of models both with the same instance component, $\langle S, I, Ext_{S,I} \rangle$ and $\langle S', I, Ext_{S',I} \rangle$. By Definition 2, for every scheme $n \in Nodes \cup Edges$, $Ext_{S,I}(n) = expr_n$ for some expression $expr_n$ in L over I. Also, for every set $i \in I$, $i = expr_i$ for some expression $expr_i$ in L over $Range(Ext_{S',I})$. Thus, every set of $Range(Ext_{S,I})$ can be derived from $Range(Ext_{S',I})$ by means of an expression in L. By a similar argument, every set of $Range(Ext_{S',I})$ can be derived from $Range(Ext_{S,I})$.

To illustrate, the top half of Figure 1 shows a schema S consisting of two nodes person and dept and an edge between them, an 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 extension mapping $Ext_{S,I}$ defined as follows:

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

The bottom half of Figure 1 shows another schema S' consisting of three nodes person, $\mathsf{works\_in}$ and dept, two edges between them, and the constraint stating that each instance of $\mathsf{works\_in}$ is connected to precisely one instance of person and dept. 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:

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

Conversely, we can define $Ext_{S,I}$ in terms of $Ext_{S',I}$ as follows (where $\bowtie$ is the natural join operator):

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

Thus S and S' are u-equivalent. We will see this u-equivalence again later, expressed at a higher level of abstraction as the entity/relationship equivalence of Figure 5(b).

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

Definition 5 Given a condition, f, $Inst(S, f)$ denotes 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 $Inst(S, f) = Inst(S', f)$.

To illustrate, in Figure 2 the schema S and the instance I are as in Figure 1. The schema S' now consists of three nodes person, $\mathsf{mathematician}$ and $\mathsf{computer\ scientist}$, with constraints stating that the last two are subsets of the first. I can be shown to be an instance of S' only if

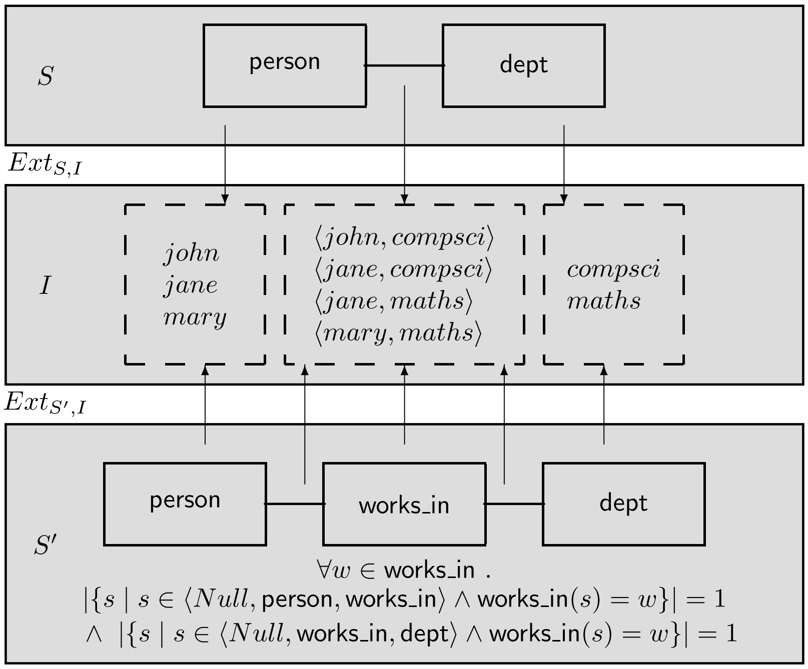


Figure 1: Two u-equivalent schemas

the domain of the dept node consists of two values. In our example this is indeed the case, the two values being *compsci* and *maths*, and we can define $Ext_{S',I}$ in terms of $Ext_{S,I}$ as follows:

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

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

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

Thus S and S' are c-equivalent with respect to the condition that $\mid Ext_{S,I}(\mathsf{dept}) \mid = 2$. We will see this c-equivalence again later, expressed as the mandatory attribute and total generalisation equivalence in Figure 3(a).

3 Transformation of Models

In this section we use the definitions of u-equivalence and c-equivalence above as the semantic foundation for defining a set of primitive transformations on models. A primitive transformation may always be applicable to a schema irrespective of the instance — in which case we call it a

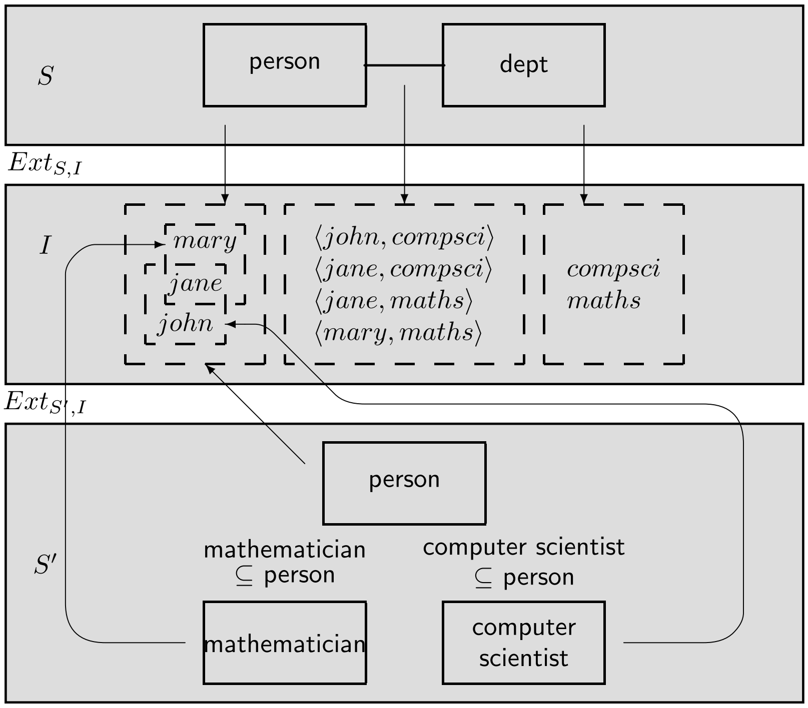


Figure 2: Two c-equivalent schemas

schema-dependent (s-d) transformation — or may only be applicable if the instance satisfies certain conditions — in which case we call it an **instance-dependent** (i-d) transformation. We show that if a schema S can be transformed to a schema S' by means of an s-d primitive transformation, 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 the expressiveness of primitive transformations by allowing them to take an extra parameter. This encodes a user-defined condition on the model which must be satisfied in order for the transformation to be applicable — we call such transformations **knowledge-based** (k-b) ones. We finally extend the treatment to composite transformations consisting of a sequence of primitive transformations.

3.1 Primitive transformations

Each primitive transformation takes a model and a further parameter and returns a new model *i.e.* it is a function of type $ArgType \rightarrow Models \rightarrow Models$ for some type $ArgType$. The instance component of the input model is left unchanged by every primitive transformation; only the schema component and the extension mapping are changed. A primitive transformation is **successful** if, were it applied, it would result in a model. If not successful, the transformation is assumed to return an "undefined" value, denoted by ϕ. The result of applying a primitive transformation to ϕ is assumed to be ϕ.

We list our primitive transformations in Definition 6 below, giving their name and the type of their first argument. Formal definitions of these transformations are given in the Appendix. In Definition 6, the type $Queries$ denotes the set of **queries** expressible in L where, given a schema $S = \langle Nodes, Edges, Constraints \rangle$ 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 set of variables, $VARS(q)$, is a subset of $Nodes \cup Edges$. If $VARS(q) = \{v_1, \ldots, v_n\}$, the **value** of q is given by $q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$. Such queries will be applied to items of the input schema and will return a set which is the extent of a new item. Thus, as with our notions of u-equivalence and c-equivalence, transformations which add new items to a schema are language-dependent.

Definition 6 The following are the primitive transformations:

1. $renameNode(Names \times Names)$ renames a node. It is successful provided either (a) the new name is not already the name of a node in the schema, or (b) a node already exists with the new name and has the same extent as the source node.

2. $renameEdge(Schemes \times Names)$ renames an edge. It is successful provided either (a) the new name is not already the name of an edge in the schema, or (b) an edge already exists with the new name and has the same extent as the source edge.

3. $addConstraint(Constraints)$ adds a new constraint, and is successful provided $Ext_{S,I}$ satisfies the new constraint.

4. $delConstraint(Constraints)$ deletes a constraint and is always successful.

5. $addNode(Names \times Queries)$ adds a new node whose extent is given by the value of the query. This is successful provided either (a) a node of that name does not already exist, or (b) a node of that name already exists with precisely the given extent.

6. $delNode(Names)$ deletes a node if it exists and participates in no edges, otherwise it has no effect on the schema. In the former case it is successful provided property (ii) of

Definition 2 is not violated by setting the extent of the node to be undefined. In the latter case it is trivially successful.

7. $addEdge(Seq(Schemes) \times Queries)$ adds a new edge between a sequence of existing schemes. The extent of the edge is given by the value of the query. This transformation is successful provided either (a) the edge does not already exist, the participating schemes exist, and the extent of the edge satisfies the appropriate domain constraints, or (b) the edge already exists with precisely the given extent.

8. $delEdge(Seq(Schemes))$ removes an edge if it exists and participates in no edges, otherwise it has no effect on the schema. In the former case it is successful provided property (ii) of Definition 2 is not violated by setting the extent of the edge to be undefined. In the latter case it is trivially successful.

We note that every primitive transformation is well-defined *i.e.* when applied to any model it yields either ϕ or another model. We also note that the primitive transformations are syntactically complete, in the sense that without their associated provisos they could be used to transform any schema into any other schema. With the addition of the provisos, the transformations become semantically sound *i.e.* they output a model as defined in Definition 3.

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 A primitive transformation t is **schema-dependent (s-d) w.r.t. a schema** S if t does not return ϕ 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 s-d primitive transformation, and vice versa, then S and S' are u-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 an i-d primitive transformation with proviso f, and vice versa, then S and S' are c-equivalent w.r.t f.

For example, if S consists of one node, employee, and S' consists of one node staff, then the transformation $rename(\mathsf{employee}, \mathsf{staff})$ on S is s-d as is the transformation $rename(\mathsf{staff}, \mathsf{employee})$ on S', and so S and S' are u-equivalent.

On the other hand, if S consists of two nodes, employee and staff, and S' consists of one node staff, then the transformation $rename(\mathsf{employee}, \mathsf{staff}$ on S is i-d with proviso that $Ext_{S,I}(\mathsf{employee}) = Ext_{S,I}(\mathsf{staff})$ while the transformation $addNode(\mathsf{employee}, \mathsf{staff})$ on S' is s-d. So overall S and S' are c-equivalent w.r.t. the condition $Ext_{S,I}(\mathsf{employee}) = Ext_{S,I}(\mathsf{staff})$.

3.2 Knowledge-based transformations

For each of the primitive transformations of Definition 6 we can 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:

$$op\ arg\ c\ m \quad = \quad \text{if } c(m) \text{ then } (op\ arg\ m) \text{ else } \phi$$

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

3.3 Composite transformations

The above treatment generalises to **composite transformations**. A composite transformation is a sequence of $n \geq 1$ primitive transformations:

$$op_1\ arg_1\ c_1;\ op_2\ arg_2\ c_2;\ \ldots;\ op_n\ arg_n\ c_n$$

where the conditions c_i are optional. If any one of this sequence of primitive transformations is not successful, *i.e.* returns ϕ, then so does the composite transformation overall. Thus if the primitive transformations have associated provisos $f_1, \ldots, f_n$ respectively, the composite transformation has the following overall proviso, where m is the model that the transformation is being applied to:

$$f_i \text{ holds for } op_{i-1}\ arg_{i-1}\ c_{i-1}\ (\ldots(op_2\ arg_2\ c_2\ (op_1\ arg_1\ c_1\ m))\ldots), \text{ for all } 1 \leq i \leq n$$

Any composite transformation T is well-defined, by virtue of the fact that the primitive transformations are so, and for input models with the same schema its output models also all have the same schema. If T's proviso holds for all models of a schema S, then T is schema-dependent (s-d) w.r.t. S. Otherwise T is instance-dependent (i-d) w.r.t. S. Notice that for T to be s-d, its first primitive transformation must individually be s-d but the remaining $n-1$ ones need not be.

As for primitive transformations, if a schema S can be transformed to a schema S' by means of a s-d composite transformation, and vice versa, then S and S' are u-equivalent. Similarly, if S can be transformed to schema S' by means of an i-d or k-b composite transformation with proviso f, and vice versa, then S and S' are c-equivalent w.r.t f.

To illustrate, for the schemas shown in Figure 1 the following composite s-d transformation will transform S to S':

$addNode$ $\langle \mathsf{works\_in}, \langle Null, \mathsf{person}, \mathsf{dept}\rangle\rangle$;
$addEdge$ $\langle Null, \mathsf{person}, \mathsf{works\_in}, \mathsf{works\_in}\rangle$;
$addEdge$ $\langle Null, \mathsf{works\_in}, \mathsf{dept}, \mathsf{works\_in}\rangle$;
$delEdge$ $\langle Null, \mathsf{person}, \mathsf{dept}\rangle$

The reverse transformation, removing the node works_in, is achieved by the following s-d transformation:

$addEdge$ $\langle Null, \mathsf{person}, \mathsf{dept}, \{\langle \mathsf{person}(s_1), \mathsf{works\_in}(s_1), \mathsf{dept}(s_2)\rangle \mid$
$s_1 \in \langle Null, \mathsf{person}, \mathsf{works\_in} \wedge s_2 \in \langle Null, \mathsf{works\_in}, \mathsf{dept}\rangle\rangle \wedge$
$\mathsf{works\_in}(s_1) = \mathsf{works\_in}(s_2)\}\rangle$;
$delEdge$ $\langle Null, \mathsf{person}, \mathsf{works\_in}\rangle$;
$delEdge$ $\langle Null, \mathsf{works\_in}, \mathsf{dept}\rangle$;
$delNode$ works_in

Thus S and S' in Figure 1 are u-equivalent.

For the schemas shown in Figure 2, the following composite transformation will transform S into S':

$addNode$ $\langle \mathsf{mathematician}, \{x \mid \langle x, maths\rangle \in \langle Null, \mathsf{person}, \mathsf{dept}\rangle\}\rangle$;
$addNode$ $\langle \mathsf{computer\ scientist}, \{x \mid \langle x, compsci\rangle \in \langle Null, \mathsf{person}, \mathsf{dept}\rangle\}\rangle$;
$addConstraint$ $(\mathsf{mathematician} \subseteq \mathsf{person})$;
$addConstraint$ $(\mathsf{computer\ scientist} \subseteq \mathsf{person})$;
$delEdge$ $\langle Null, \mathsf{person}, \mathsf{dept}\rangle$ $(\mathsf{dept} = \{maths, compsci\})$;
$delNode$ dept

Note the condition on the last-but-one primitive transformation, making the composite transformation k-b overall. By contrast, the reverse transformation is s-d:

$addNode$ $\langle \mathsf{dept}, \{maths, compsci\}\rangle$;
$addEdge$ $\langle Null, \mathsf{person}, \mathsf{dept}, \{\langle x, maths\rangle \mid x \in Ext_{S',I}(\mathsf{mathematician})\} \cup \{\langle x, compsci\rangle \mid x \in \mathsf{computer\,scientist}\}\rangle$;
$delConstraint$ $(\mathsf{mathematician} \subseteq \mathsf{person})$;
$delConstraint$ $(\mathsf{computer\ scientist} \subseteq \mathsf{person})$;
$delNode$ $\mathsf{mathematician}$;
$delNode$ $\mathsf{computer\ scientist}$

Thus S and S' in Figure 2 are c-equivalent.

4 Expressiveness of the approach

A practical CDM will have higher-level constructs than nodes, edges and constraints. Thus appropriate composite transformations will be required in order to transform schemas expressed in such a CDM, and these can be built up from the primitive transformations that we defined above. In this section we illustrate how a higher-level CDM and transformations on it can be defined by first showing how to define the binary ER schemas and primitive transformations on them that we gave in [11]. We further demonstrate the applicability of our framework by extending this treatment to a much richer ER CDM that supports n-ary relations, attributes on relations, complex attributes, and generalisation hierarchies.

4.1 Transformations for a binary ER CDM

The following definition of a binary ER schema is as in [11]:

Definition 8 A **binary 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 the directed graph induced by $Incs$ 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 and c_2 are both of the form $l : u$ where l is a natural number and u is either a natural number or N (denoting no upper limit). 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.

(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 notice that entity names and attribute names are unique, and that an entity and an attribute cannot have the same name (because $Ents \cup Atts$ corresponds to *Nodes* in the underlying hypergraph). However, attributes can be shared between entities and relationships. *Assocs* corresponds to *Edges* and *Incs* to *Constraints* in the underlying hypergraph.

We next define a set of transformations on binary ER schemas in terms of the primitive transformations we gave in Section 3.1. We will use some short-hand notation for expressing cardinality constraints on associations. Although for the moment only binary associations are necessary, we anticipate the need for n-ary ones in Section 4.2. Thus, we denote by $makeCard\ \langle n_0, n_1, \ldots, n_m, l_1 : u_1, \ldots, l_m : u_m \rangle$ the following cardinality constraint on the m-ary scheme $\langle n_0, n_1, \ldots, n_m \rangle$:

$$\bigwedge_{i=1}^{m} (\forall s_i \in n_i \,.\, l_i \leq |\{s | s \in \langle n_0, n_1, \ldots, n_m \rangle \wedge n_i(s) = s_i\}| \leq u_i)$$

Conversely, we denote by $getCard\ \langle n_0, n_1, \ldots, n_m \rangle$ the cardinality constraint associated with the scheme $\langle n_0, n_1, \ldots, n_m \rangle$ *i.e.* the above conjunction.

The primitive transformations on binary ER schemas given in [11] can be defined as follows in terms of the primitive transformations of Section 3.1:

- $rename_E\ \langle from, to \rangle$ and $rename_A\ \langle from, to \rangle$ which respectively rename an entity type and an attribute, can both be implemented by:

  $renameNode \quad \langle from, to \rangle$

- $rename_R\ \langle from, to \rangle$ which renames a relationship can be implemented by [1]:

  $renameEdge \quad \langle from, to \rangle$

- $expand\ \langle n_0, n_1, n_2, l_1 : u_1, l_2 : u_2 \rangle$ which replaces the old cardinality constraint on the association $\langle n_0, n_1, n_2 \rangle$ by the new, relaxed, constraint $l_1 : u_1, l_2 : u_2$ is implemented by:

  $delConstraint \quad (getCard\ \langle n_0, n_1, n_2 \rangle);$
  $addConstraint \quad (makeCard\ \langle n_0, n_1, n_2, l_1 : u_1, l_2 : u_2 \rangle)$

- $contract\ \langle n_0, n_1, n_2, l_1 : u_1, l_2 : u_2 \rangle$ which replaces the old cardinality constraint on the association $\langle n_0, n_1, n_2 \rangle$ by the new, stricter, constraint $l_1 : u_1, l_2 : u_2$ is implemented similarly.

[1] There is a slight departure here from this transformation as described in [11] which took only the relationship name as its first parameter. This was not correct since two relationships can have the same name and so the entire scheme is needed to uniquely identify a relationship.

- $add_E\langle e, q\rangle$ which adds an entity type e to the schema and assigns it the extent defined by the query q:

  $addNode \quad \langle e, q\rangle$

- $del_E\ e$ which deletes entity type e if it has no attributes and participates in no relationships:

  $delNode \quad e$

- $add_R\ \langle r, e_1, e_2, l_1 : u_1, l_2 : u_2, q\rangle$ which adds the relationship $\langle r, e_1, e_2, l_1 : u_1, l_2 : u_2\rangle$ to the set of associations of the schema and assigns it the extent defined by the query q:

  $addEdge \quad \langle r, e_1, e_2, q\rangle$;
  $addConstraint \quad (makeCard\ \langle r, e_1, e_2, l_1 : u_1, l_2 : u_2\rangle)$

- $del_R\ \langle r, e_1, e_2\rangle$ which removes the relationship $\langle r, e_1, e_2\rangle$ from the set of associations of the schema:

  $delConstraint \quad (getCard\ \langle r, e_1, e_2\rangle)$;
  $delEdge \quad \langle r, e_1, e_2\rangle$

- $add_A\ \langle e, a, l_1 : u_1, l_2 : u_2, q_{att}, q_{assoc}\rangle$ which adds the association $\langle Null, e, a, l_1 : u_1, l_2 : u_2\rangle$ to the schema, assigning the attribute extent q_{att} and the association extent q_{assoc}:

  $addNode \quad \langle a, q_{att}\rangle$;
  $addEdge \quad \langle Null, e, a, q_{assoc}\rangle$;
  $addConstraint \quad (makeCard\ \langle e, a, l_1 : u_1, l_2 : u_2\rangle)$

- $del_A\ \langle e, a\rangle$ which removes the association $\langle Null, e, a\rangle$ from the set of associations of the schema:

  $delConstraint \quad (getCard\ \langle Null, e, a\rangle)$;
  $delEdge \quad \langle Null, e, a\rangle$;
  $delNode \quad a$

- $add_I\ \langle e_1, e_2\rangle$ which adds this subtype relationship to the schema, provided that the extent of e_1 is indeed contained in the extent of e_2:

  $addConstraint \quad (e_1 \subseteq e_2)$

- $del_I\ \langle e_1, e_2\rangle$ which removes this subtype relationship from the schema:

  $delConstraint \quad (e_1 \subseteq e_2)$

k-b versions of these transformations can be defined by adding the constraint to the first primitive transformation of the composition. In [11] we illustrated the expressiveness of these transformations on binary ER schemas by defining many of the common schema equivalences found in the literature and thus deriving whether they conditional or unconditional. We do not repeat this work here. Instead we define a richer ER CDM and transformations on it in Section 4.2 below. We define some equivalences for this richer model in Section 5.

4.2 Transformations for an enriched ER CDM

Our enriched ER CDM supports n-ary relations, attributes on relations, complex attributes, and generalisation hierarchies. N-ary relations are readily supported since the underlying hypergraph can have edges connecting arbitrarily many nodes. Relationships with attributes are supported since schemes can be nested within schemes (though only one level of such nesting is needed for this CDM). To support complex attributes, we extend the syntax of add_A to specify a path starting at an entity or relationship and ending with the new, possibly nested, attribute. del_A is similarly generalised. Set-valued attributes are already the default since the cardinality of attributes is constrained by additional cardinality constraints as required. Finally we need one more set, $Total = \{\mathsf{partial}, \mathsf{total}\}$, in order to indicate whether a generalisation is partial or total [9].

Definition 9 An **enriched ER schema**, S, is a quadruple $\langle Ents, Gens, Atts, Assocs\rangle$ where:

- $Ents \subseteq Names$ is the set of entity-type names.
- $Gens \subseteq Total \times Seq(Names)$ is the set of generalisations. There is a tuple in $Gens$ of the form $\langle t, e, e_1, \ldots, e_n\rangle$ if entity type e is a generalisation of entity types $e_1, \ldots, e_n$. The generalisation is partial/total according to the value of t. We assume that the directed graph induced by $Gens$ is acyclic.
- $Atts \subseteq Names$ is the set of attribute names.
- $Assocs \subseteq Names \times Seq(Schemes) \times Seq(Cards)$ is the set of **associations**, where:

  (i) For each relationship between n entity types $e_1, \ldots, e_n \in Ents$, there is a tuple in $Assocs$ of the form:
  $$\langle rel\_name, e_1, \ldots, e_n, c_1, \ldots, c_n\rangle$$
  c_i indicates the lower and upper cardinalities of participations in the relationship by each instance of e_i. rel_name may be $Null$ if there is only one relationship between $e_1, \ldots, e_n$.

  (ii) For each attribute a associated with an entity type e there is a tuple in $Assocs$ of the form:
  $$\langle Null, e, a, c_1, c_2\rangle$$

  (iii) For each attribute a associated with a relationship $\langle r, e_1, \ldots, e_n\rangle$ there is a tuple in $Assocs$ of the form:
  $$\langle Null, \langle r, e_1, \ldots, e_n\rangle, a, c_1, c_2\rangle$$

  (iv) For each sub-attribute b of a parent attribute a there is a tuple in $Assocs$ of the form:
  $$\langle Null, a, b, c_1, c_2\rangle$$

The primitive transformations on these enriched ER schemas are defined as follows in terms of the primitive transformations of Section 3.1:

- $rename_X$ $\langle from, to\rangle$ where X can be E, A or R is implemented by $renameNode$ or $renameEdge$, as for binary ER schemas in Section 4.1 above.

- *expand* $\langle n_0, n_1, \ldots, n_m, l_1 : u_1, \ldots, l_m : u_m\rangle$ which replaces the old cardinality constraint on the association $\langle n_0, n_1, \ldots, n_m\rangle$ by the new, relaxed, constraint $l_1 : u_1, \ldots, l_m : u_m$ is implemented by

  $delConstraint \quad (getCard\ \langle n_0, n_1, \ldots, n_m\rangle);$
  $addConstraint \quad (makeCard\ \langle n_0, n_1, \ldots, n_m, l_1 : u_1, \ldots, l_m : u_m\rangle)$

- *contract* $\langle n_0, n_1, \ldots, n_m, l_1 : u_1, \ldots, l_m : u_m\rangle$ which replaces the old cardinality constraint on the association $\langle n_0, n_1, \ldots, n_m\rangle$ by the new, stricter, constraint $l_1 : u_1, \ldots, l_m : u_m$ is implemented similarly.

- $add_E\ \langle e, q\rangle$ which adds an entity type e to the schema and assigns it the extent defined by the query q:

  $addNode \quad \langle e, q\rangle$

- $del_E\ e$ which deletes an entity type e if it has no attributes and participates in no relationships:

  $delNode \quad e$

- $add_R\ \langle r, e_1, \ldots, e_n, l_1 : u_1, \ldots, l_n : u_n, q\rangle$ which adds this relationship to the set of associations of the schema and assigns it the extent defined by the query q:

  $addEdge \quad \langle r, e_1, \ldots, e_n, q\rangle;$
  $addConstraint \quad (makeCard\ \langle r, e_1, \ldots, e_n, l_1 : u_1, \ldots, l_n : u_n\rangle)$

- $del_R\ \langle r, e_1, \ldots, e_n\rangle$ which removes this relationship from the set of associations of the schema:

  $delConstraint \quad (getCard\ \langle r, e_1, \ldots, e_n\rangle);$
  $delEdge \quad \langle r, e_1, \ldots, e_n\rangle$

- $add_A\ \langle a_0, a_1, \ldots, a_n, l_1 : u_1, l_2 : u_2, q_{att}, q_{assoc}\rangle$, where a_0 is an entity type or relationship and $n \geq 1$, adds the association between a_{n-1} and a_n to the schema, assigning the attribute a_n extent q_{att} and the association extent q_{assoc}:

  $addNode \quad \langle a_n, q_{att}\rangle;$
  $addEdge \quad \langle Null, a_{n-1}, a_n, q_{assoc}\rangle;$
  $addConstraint \quad (makeCard\ \langle Null, a_{n-1}, a_n, l_1 : u_1, l_2 : u_2\rangle)$

- $del_A\ \langle a_0, a_1, \ldots, a_n\rangle$ which removes the association $\langle Null, a_{n-1}, a_n\rangle$ from the set of associations of the schema:

  $delConstraint \quad (getCard\ \langle Null, a_{n-1}, a_n\rangle);$
  $delEdge \quad \langle Null, a_{n-1}, a_n\rangle;$
  $delNode \quad a_n$

- $add_G\ \langle \mathsf{partial}, e, e_1, \ldots, e_n\rangle$ which adds this generalisation to the schema, provided that the extents of $e_1, \ldots, e_n$ are disjoint and contained within the extent of e:

  $addConstraint \quad \forall 1 \leq i \leq n\ .\ e_i \subseteq e;$
  $addConstraint \quad \forall 1 \leq i < j \leq n\ .\ e_i \cap e_j = \emptyset$

- add_G $\langle \mathsf{total}, e, e_1, \ldots, e_n \rangle$ is equivalent to add_G $\langle \mathsf{partial}, e, e_1, \ldots, e_n \rangle$ with the additional constraint that $e_1, \ldots, e_n$ completely cover e:

  $addConstraint$ $\forall 1 \leq i \leq n \,.\, e_i \subseteq e;$
  $addConstraint$ $\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset;$
  $addConstraint$ $e = \bigcup_{i=1}^{n} e_i$

- del_G $\langle \mathsf{partial}, e, e_1, \ldots, e_n \rangle$ removes this generalisation from the schema by removing the constraints it implies:

  $delConstraint$ $\forall 1 \leq i \leq n \,.\, e_i \subseteq e;$
  $delConstraint$ $\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset$

- del_G $\langle \mathsf{total}, e, e_1, \ldots, e_n \rangle$ similarly removes the constraints this generalisation implies:

  $delConstraint$ $\forall 1 \leq i \leq n \,.\, e_i \subseteq e;$
  $delConstraint$ $\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset;$
  $delConstraint$ $e = \bigcup_{i=1}^{n} e_i$

5 Some Example Equivalences

In this section we demonstrate our approach by defining, and thereby formalising, a number of equivalences on enriched ER schemas that have appeared in the literature. To aid presentation we have grouped these equivalences into three subsections according to what schema constructs are being equated. Figures 1 - 3 graphically illustrate these three sets of equivalences. In these figures a shaded hexagon indicates a total generalisation while a blank hexagon indicates a partial one. Each equivalence is illustrated both generally and by a specific example on the same figure.

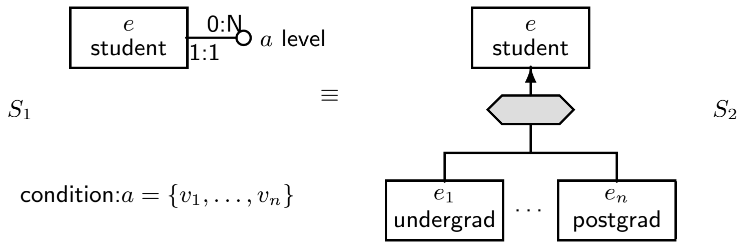
5.1 Equivalences involving generalisations and attributes

Figure 3 illustrates three equivalences between pairs of schemas S_1 and S_2 involving attributes and generalisations. The first equivalence has been formalised previously in [5], and illustrates how our formalism differs from that approach. The second and third equivalences illustrate how two different equivalences arise when we formalise the "attribute moving" operations found in a number of papers [2, 9, 4], which have not considered the cardinalities of attributes.

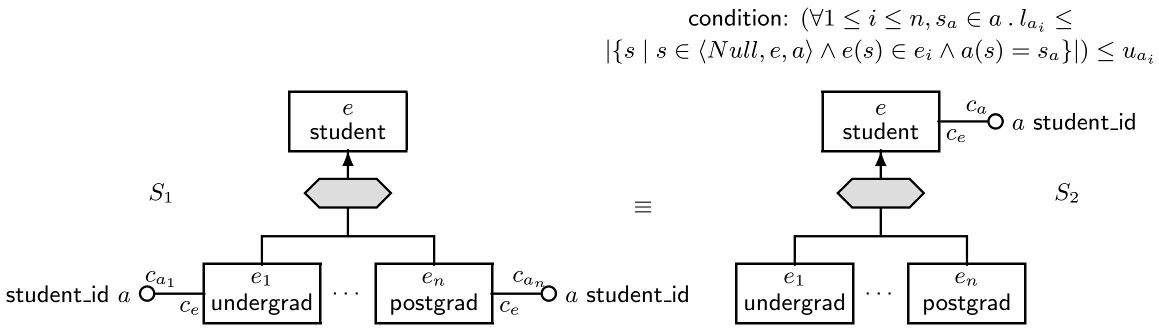
Mandatory attribute and total generalisation equivalence. This exists between two schemas S_1 and S_2 when S_1 contains an association between an entity type e and an attribute a with cardinality constraints $1{:}1$ and $0{:}N$ and S_2 contains a total generalisation (see Figure 3(a)). S_1 is transformed to S_2 as follows:

add_E $\langle e_1, \{e(s) \mid s \in \langle Null, e, a \rangle \wedge a(s) = v_1\} \rangle;$
$\vdots$
add_E $\langle e_n, \{e(s) \mid s \in \langle Null, e, a \rangle \wedge a(s) = v_n\} \rangle;$
add_G $\langle \mathsf{total}, e, e_1, \ldots, e_n \rangle;$
del_A $\langle e, a \rangle$ $(a = \{v_1, \ldots, v_n\})$

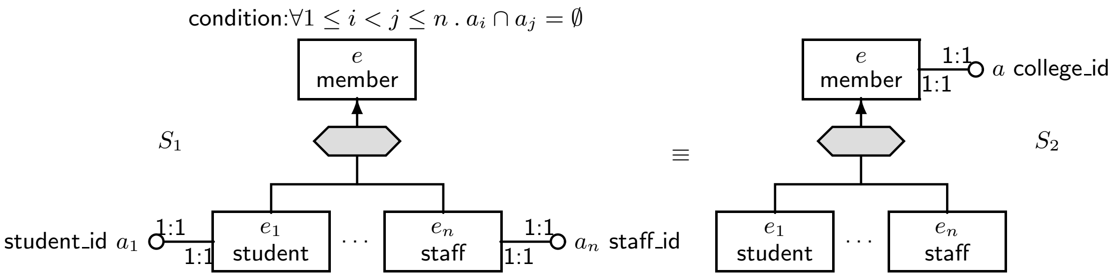
Note that the last step is a k-b transformation (*i.e.* one with a condition), making the whole transformation k-b. Intuitively we can only delete attribute a if its extent consists of the values $v_1, \ldots, v_n$ that were used to determine the extents of $e_1, \ldots, e_n$.



(a) Mandatory attribute and total generalisation equivalence



(b) Attribute generalisation



(c) Key attribute generalisation

Figure 3: Equivalences involving generalisations and attributes

The reverse transformation from S_2 to S_1 is s-d:

$add_A \quad \langle e, a, 1{:}1, 0{:}N, \{v_1, \ldots, v_n\}, \cup_{i=1}^{n}\{\langle s, v_i\rangle \mid s \in e_i\}\rangle;$
$del_G \quad \langle \mathsf{total}, e, e_1, \ldots, e_n\rangle;$
$del_E \quad e_1;$
$\vdots$
$del_E \quad e_n$

Thus the two schemas are c-equivalent with respect to the condition $Ext_{S_1,I}(a) = \{v_1, \ldots, v_n\}$. A specific instance of the equivalence is illustrated in Figure 3(a), where S_1 contains a student entity type with an attribute level that takes one of two values, postgrad and undergrad (so $n = 2$ here), and S_2 has a generalisation student of two entity types postgrad and undergrad.

We note that replacing in Figure 3(a) the total generalisation in S_2 by a partial one and the mandatory attribute in S_1 by an optional one (*i.e.* cardinality $0{:}1$ on e) gives the equivalence between an optional attribute and a partial generalisation. The above transformations need to be modified to replace total by partial and $1{:}1$ by $0{:}1$.

Attribute generalisation. This exists between two schemas S_1 and S_2 when in S_1 all subtypes of a generalisation share a common attribute a while in S_2 the attribute is associated with the supertype (see Figure 3(b)). S_1 is transformed to S_2 by the following s-d transformation, where if each $c_{a_i} = l_{a_i}{:}u_{a_i}$ then $c_a = \sum_{i=1}^{n} l_{a_i}{:}\sum_{i=1}^{n} u_{a_i}$:

$add_A \quad \langle e, a, c_e, c_a, a, \cup_{i=1}^{n}\langle Null, e_i, a\rangle\rangle;$
$del_A \quad \langle e_i, a\rangle;$
$\vdots$
$del_A \quad \langle e_n, a\rangle$

The reverse transformation is dependent on the associations between s and the subtypes of e satisfying the stated cardinality constraints:

$add_A \quad \langle e_1, a, c_e, c_{a_1}, a, \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_1\}\rangle$
$\quad (\forall s_a \in a \,.\, l_{a_1} \leq |\{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_1 \wedge a(s) = s_a\}| \leq u_{a_1});$
$\vdots$
$add_A \quad \langle e_n, a, c_e, c_{a_n}, a, \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_n\}\rangle$
$\quad (\forall s_a \in a \,.\, l_{a_n} \leq |\{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_n \wedge a(s) = s_a\}| \leq u_{a_n});$
$del_A \quad \langle e, a\rangle$

Thus the two schemas are c-equivalent w.r.t. the stated conditions. An instance of the equivalence is illustrated in Figure 3(b), where in S_1 student_id is an attribute of postgrad and undergrad, with $c_{a_1} = c_{a_2} = 0 : 1$. Moving the attribute to student in S_2 gives $c_a = 0 : 2$. The reverse transformation requires each student no. to be associated with no more than one postgrad and no more than one undergrad.

Key attribute generalisation. In contrast to attribute generalisation, this involves merging distinct key attributes on subtypes ($a_1, \ldots, a_n$ in Figure 3(c)) to form a single key attribute on the supertype (a). S_1 is transformed to S_2 as follows:

$add_A \quad \langle e, a, 1{:}1, 1{:}1, \cup_{i=1}^{n} a_i, \cup_{i=1}^{n}\langle e_i, a_i\rangle\rangle \quad (\forall 1 \leq i < j \leq n \,.\, a_i \cap a_j = \emptyset);$
$del_A \quad \langle e_i, a_i\rangle;$
$\vdots$
$del_A \quad \langle e_n, a_n\rangle$

The reverse transformation is s-d:

$add_A \quad \langle e_1, a_1, 1{:}1, 1{:}1, \{a(s) \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_1\},$
$\qquad \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_1\}\rangle;$

$\vdots$

$add_A \quad \langle e_n, a_n, 1{:}1, 1{:}1, \{a(s) \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_n\},$
$\qquad \{s \mid s \in \langle Null, e, a\rangle \wedge e(s) \in e_n\}\rangle;$

$del_A \quad \langle e, a\rangle$

Thus the two schemas are c-equivalent with respect to the condition $\forall 1 \leq i < j \leq n \,.\, Ext_{S_1,I}(a_i) \cap Ext_{S_1,I}(a_j) = \emptyset$. A specific instance of the equivalence is illustrated in Figure 3(c) where in S_1 student_id identifies student instances and staff_id identifies staff instances. In a schema improvement process, we might want to merge these to form a single key attribute college_id on the generalisation member (meaning member of the college), as in S_2. This requires the additional knowledge that the extents of student_id and staff_id do not intersect. The reverse transformation is independent of the instance, since we may always partition the set of keys for a supertype into keys for its subtypes.

5.2 Equivalences between generalisations

Figure 4 illustrates three of the equivalences proposed in [9], where they are used as part of a methodology for integrating generalisation hierarchies.

Introduction of total generalisation. This exists between two schemas S_1 and S_2 when S_1 contains a set of entity types with distinct extents and S_2 contains a generalisation entity type whose extent is the union of these (see Figure 4(a)). S_1 is transformed to S_2 by the following k-b transformation:

$add_E \quad \langle e, \cup_{i=1}^{n} e_i\rangle \qquad (\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset);$

$add_G \quad \langle \mathsf{total}, e, e_1, \ldots, e_n\rangle$

The reverse transformation is s-d, since we can always recover e by forming the union of $e_1, \ldots, e_n$:

$del_G \quad \langle \mathsf{total}, e, e_1, \ldots, e_n\rangle;$

$del_E \quad e$

Thus the two schemas are c-equivalent with respect to the condition $\forall 1 \leq i < j \leq n \,.\, Ext_{S_1,I}(e_i) \cap Ext_{S_1,I}(e_j) = \emptyset$. A specific instance of the equivalence is illustrated in Figure 4(a), where in S_1 the entity types undergrad and postgrad are known to be disjoint and in S_2 there is a total generalisation student of these.

Identification of total generalisation. Here S_1 contains entity types $e_1, \ldots, e_n$ with disjoint extents and a partial generalisation, e, thereof while S_2 contains an extra total generalisation, e_s, of $e_1, \ldots, e_n$ (see Figure 4(b)). S_1 is transformed to S_2 by the following s-d transformation:

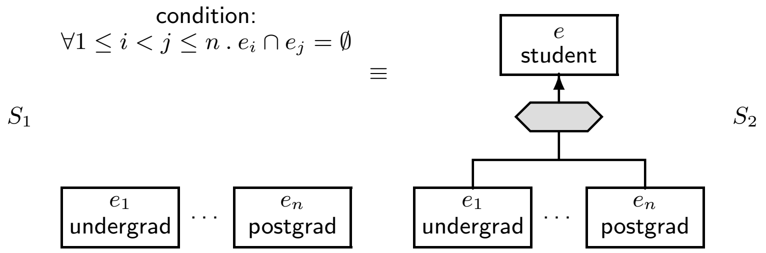
$add_E \quad \langle e_s, \cup_{i=1}^{n} e_i\rangle;$

$add_G \quad \langle \mathsf{partial}, e, e_s\rangle;$

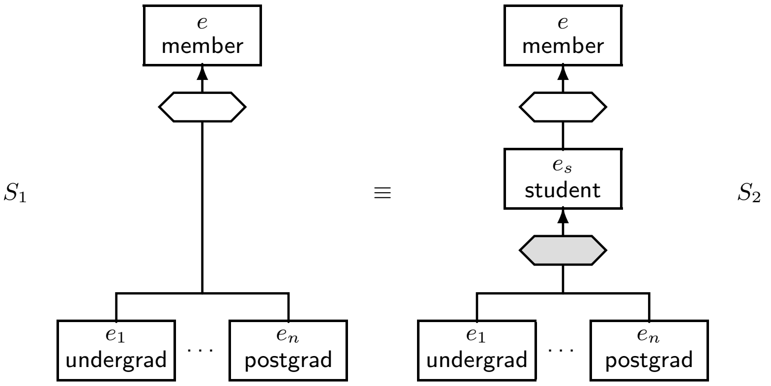
$del_G \quad \langle \mathsf{partial}, e, e_1, \ldots, e_n\rangle;$

$add_G \quad \langle \mathsf{total}, e_s, e_1, \ldots, e_n\rangle$

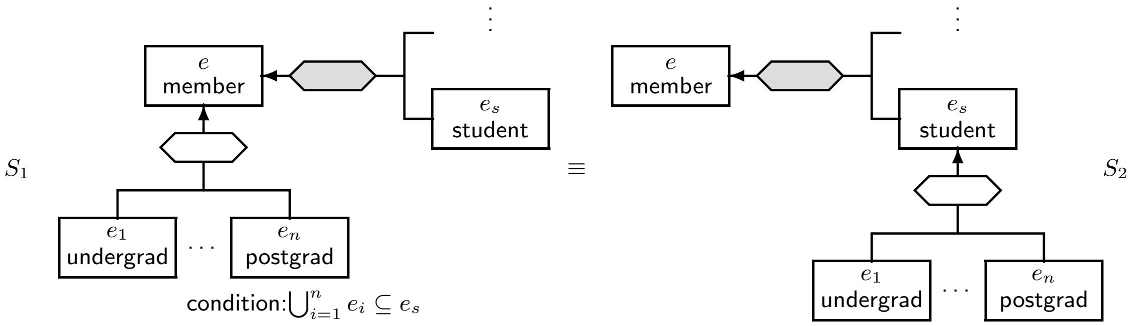
The reverse transformation is s-d, since we can always recover e_s by forming the union of $e_1, \ldots, e_n$:



(a) Introduction of total generalisation



(b) Identification of total generalisation



(c) Move generalisation

Figure 4: Equivalences between generalisations

$del_G \quad \langle \mathsf{total}, e_s, e_1, \ldots, e_n \rangle;$
$add_G \quad \langle \mathsf{partial}, e, e_1, \ldots, e_n \rangle;$
$del_G \quad \langle \mathsf{partial}, e, e_s \rangle;$
$del_E \quad e_s$

Thus the two schemas are u-equivalent. A specific instance of the equivalence is illustrated in Figure 4(b) where the knowledge that the extents of undergrad and postgrad are disjoint is recorded in the schema by the partial generalisation member, and hence we can always introduce the intermediate student entity type.

Move generalisation. This exists between two schemas S_1 and S_2 when S_1 has a partial generalisation e of $e_1, \ldots, e_n$, and these are all subtypes of some other specialisation e_s of e (see Figure 4(c)): S_1 is transformed to S_2 by the following i-d transformation (i-d because of the implicit proviso on add_G that $\forall 1 \leq i \leq n \, . \, e_i \subseteq e_s$):

$del_G \quad \langle \mathsf{partial}, e, e_1, \ldots, e_n \rangle;$
$add_G \quad \langle \mathsf{partial}, e_s, e_1, \ldots, e_n \rangle$

The reverse transformation is clearly s-d, intuitively because it is always possible to move a generalisation of $e_1, \ldots, e_n$ up the hierarchy, reducing the constraints on the extents of these entity types:

$del_G \quad \langle \mathsf{partial}, e_s, e_1, \ldots, e_n \rangle;$
$add_G \quad \langle \mathsf{partial}, e, e_1, \ldots, e_n \rangle;$

Thus the two schemas are c-equivalent with respect to the condition $\forall 1 \leq i \leq n . Ext_{S_1,I}(e_i) \subseteq Ext_{S_1,I}(e_s)$. A specific instance of this equivalence is illustrated in Figure 4(c), where in S_1 the knowledge that undergrad and postgrad are subsets of student allows us to move undergrad and postgrad to be subtypes of student in S_2.

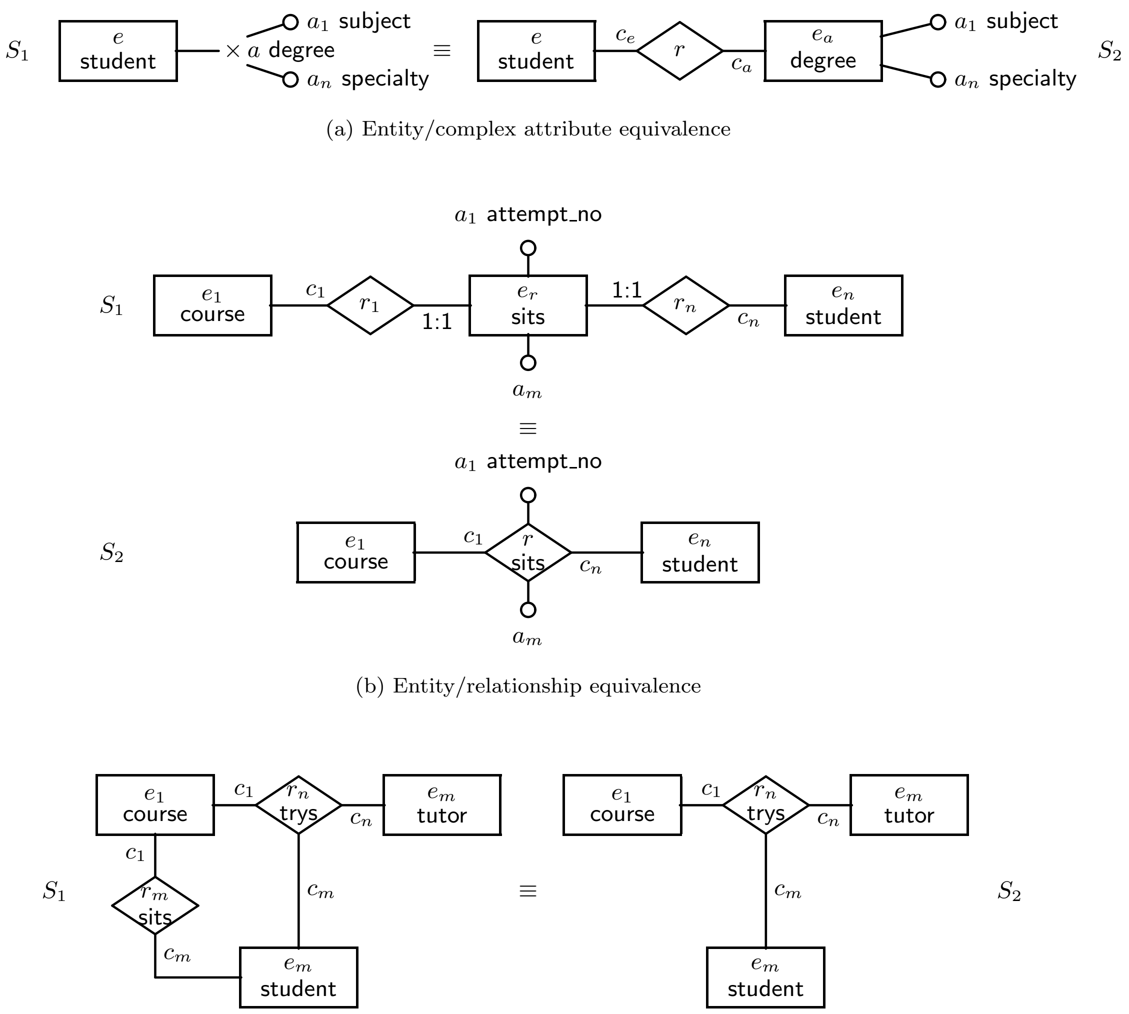
5.3 N-ary Relationships and Complex Attributes

The first two equivalences in Figure 5 were proposed in [16, 17]. The third is a new equivalence we introduce, which removes the redundancy that occurs when schemas containing relationships of varying arity are merged.

Entity/complex attribute equivalence. This exists between two schemas S_1 and S_2 when S_1 contains a complex attribute a with sub-attributes $a_1, \ldots, a_n$ and S_2 contains an entity type e_a with the same n attributes (see Figure 5(a)). S_1 is transformed to S_2 by the following s-d transformation:

$add_E \quad \langle e_a, a \rangle;$
$add_R \quad \langle r, e, e_a, c_{e,a}, c_{a,e}, \langle Null, e, a \rangle \rangle;$
$add_A \quad \langle e_a, a_1, c_{a,a_1}, c_{a_1,a}, a_1, \langle Null, a, a_1 \rangle \rangle;$
$\vdots$
$add_A \quad \langle e_a, a_n, c_{a,a_n}, c_{a_n,a}, a_n, \langle Null, a, a_n \rangle \rangle;$
$del_A \quad \langle e, a, a_1 \rangle;$
$\vdots$
$del_A \quad \langle e, a, a_n \rangle;$
$del_A \quad \langle e, a \rangle$

The reverse transformation is straightforward and is also s-d. Thus the two schemas are u-equivalent. An instance of this equivalence is illustrated in Figure 5(a), where in S_1 the complex



(a) Entity/complex attribute equivalence

(b) Entity/relationship equivalence

(c) Redundant relationship removal

Figure 5: n-ary Relationships and Complex Attributes

attribute degree consists of subject and specialty whereas in S_2 there is a degree entity with the same attributes. This equivalence would allow S_1 to be merged with another schema containing a degree entity type possibly associated with more attributes.

Entity/relationship equivalence. This exists between two schemas S_1 and S_2 when S_1 has an entity type e_r with attributes $a_1, \ldots, a_m$ and n binary relationships $r_1, \ldots, r_n$ between e_r and entity types $e_1, \ldots, e_n$, and S_2 has an n-ary relationship r between $e_1, \ldots, e_n$ with attributes $a_1, \ldots, a_m$ (see Figure 5(b)). S_1 is transformed to S_2 by the following s-d transformation:

$$
\begin{array}{ll}
add_R & \langle r, e_1, \ldots, e_n, c_1, \ldots, c_n, \{\langle e_r(s_1), e_1(s_1), \ldots, e_n(s_n)\rangle \mid \\
 & \qquad s_1 \in \langle r_1, e_1, e_r\rangle \wedge \ldots \wedge s_n \in \langle r_n, e_n, e_r\rangle \wedge e_r(s_1) = \ldots = e_r(s_n)\}\rangle; \\
add_A & \langle\langle r, e_1, \ldots, e_n\rangle, a_1, c_{e_r,a_1}, c_{a_1,e_r}, a_1, \\
 & \qquad \{\langle s, a_1(s_a)\rangle \mid s \in \langle r, e_1, \ldots, e_n\rangle \wedge s_a \in \langle Null, e_r, a_1\rangle \wedge e_r(s) = e_r(s_a)\}\rangle; \\
 & \vdots \\
add_A & \langle\langle r, e_1, \ldots, e_n\rangle, a_m, c_{e_r,a_m}, c_{a_m,e_r}, a_m, \\
 & \qquad \{\langle s, a_m(s_a)\rangle \mid s \in \langle r, e_1, \ldots, e_n\rangle \wedge s_a \in \langle Null, e_r, a_m\rangle \wedge e_r(s) = e_r(s_a)\}\rangle; \\
del_R & \langle r_1, e_1, e_r\rangle; \\
 & \vdots \\
del_R & \langle r_n, e_n, e_r\rangle; \\
del_A & \langle e_r, a_1\rangle; \\
 & \vdots \\
del_A & \langle e_r, a_m\rangle; \\
del_E & e_r
\end{array}
$$

The reverse transformation is also s-d:

$$
\begin{array}{ll}
add_E & \langle e_r, \langle r, e_1, \ldots, e_n\rangle\rangle; \\
add_A & \langle e_r, a_1, c_{r,a_1}, c_{a_1,r}, a_1, \langle Null, \langle r, e_1, \ldots, e_n\rangle, a_1\rangle\rangle; \\
 & \vdots \\
add_A & \langle e_r, a_m, c_{r,a_m}, c_{a_m,r}, a_m, \langle Null, \langle r, e_1, \ldots, e_n\rangle, a_m\rangle\rangle; \\
add_R & \langle r_1, e_r, e_1, 1:1, c_1, \langle r, e_1, \ldots, e_n\rangle\rangle; \\
 & \vdots \\
add_R & \langle r_n, e_r, e_n, 1:1, c_n, \langle r, e_1, \ldots, e_n\rangle\rangle; \\
del_A & \langle\langle r, e_1, \ldots, e_n\rangle, a_1\rangle; \\
 & \vdots \\
del_A & \langle\langle r, e_1, \ldots, e_n\rangle, a_m\rangle; \\
del_R & \langle r, e_1, \ldots, e_n\rangle;
\end{array}
$$

Thus the two schemas are u-equivalent. An instance of this equivalence is illustrated in Figure 5(b), where in S_1 the entity type sits represents a student's attempt to pass a course and the attribute attempt_no stores which attempt this is (first, second etc.). In S_2 this information is instead represented by the relationship sits with attribute attempt_no.

Redundant relationship removal. This exists when an m-ary relationship (such as r_m in Figure 5(c)) is a projection of an n-ary relationship (such as r_n), where $m \leq n$. r_m may be removed by the following k-b transformation:

$$
\begin{array}{lll}
del_R & \langle r_m, e_1, \ldots, e_m\rangle & (\{\langle e_1(s), \ldots, e_m(s)\rangle \mid s \in \langle r_m, e_1, \ldots, e_m\rangle\} = \\
 & & \{\langle e_1(s), \ldots, e_m(s)\rangle \mid s \in \langle r_n, e_1, \ldots, e_n\rangle\})
\end{array}
$$

Name	Figure	$S_1 \rightarrow S_2$	$S_1 \leftarrow S_2$	$S_1 \equiv S_2$
Mandatory attribute and total generalisation equivalence	3(a)	k-b	s-d	c
Attribute generalisation	3(b)	s-d	k-b	c
Key attribute generalisation	3(c)	k-b	s-d	c
Introduction of total generalisation	4(a)	k-b	s-d	c
Identification of total generalisation	4(b)	s-d	s-d	u
Move generalisation	4(c)	i-d	s-d	c
Entity/complex attribute equivalence	5(a)	s-d	s-d	u
Entity/relationship equivalence	5(b)	s-d	s-d	u
Redundant relationship removal	5(c)	k-b	s-d	c

Table 1: Summary of Equivalences

The reverse transformation is s-d, since it is always possible to project out some participant entities in a relationship:

$add_R \quad \langle r_m, e_1, \ldots, e_m, c_1, \ldots, c_m, \{\langle e_1(s), \ldots, e_m(s)\rangle \mid s \in \langle r_n, e_1, \ldots, e_n\rangle\}\rangle$

Thus the two schemas are c-equivalent w.r.t. the condition

$$\{\langle e_1(s), \ldots, e_m(s)\rangle \mid s \in Ext_{S_1,I}(\langle r_m, e_1, \ldots, e_m\rangle)\} = \\ \{\langle e_1(s), \ldots, e_m(s)\rangle \mid s \in Ext_{S_1,I}(\langle r_n, e_1, \ldots, e_n\rangle)\}$$

The equivalence is illustrated in Figure 5(c), where in S_1 we have a 3-ary relationship trys between course, student and tutor indicating the tutor allocated to each student trying a course. There is also a redundant 2-ary relationship sits which is the projection of trys onto course and student. In a schema improvement process, we may remove such a redundant relationship to give the schema S_2 containing just the trys relationship.

5.4 Summary

Table 1 summarises the equivalences that we have considered in this section. The columns headed $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$ state whether the left-to-right and right-to-left transformations are s-d, i-d or k-b. The column headed $S_1 \equiv S_2$ states whether the equivalence is unconditional or conditional.

6 Related Work

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**. In practice, schema conforming transformations are applied bi-directionally and schema merging and restructuring ones uni-directionally. However, in all cases there is the underlying notion that the schema is being transformed to an equivalent one (at least for some instances of the database). For each transformation, the original and resulting schema obey one or more alternative notions of schema **equivalence** [14, 1, 8], which basically vary in the mapping rules relating elements of the two schemas. This paper has presented a unifying formalism for the schema transformation process by defining a very general notion of schema equivalence, together with a set of primitive transformations that can be used to formally define more complex schema transformations.

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 [12, 13], or is limited to certain types of transformation only [3, 8, 17, 5]. The latter cases assume that specific types of dependency constraints are employed to limit the instances of schemas (or "real world states" [8]) 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. Thus, constructing transformations is a relatively simple task of programming a sequence of primitive transformations, stating conditions on these where they are dependent on instances satisfying certain constraints in order to output a valid model. 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.

A further distinctive feature of the work described here is that our underlying CDM is a very simple one. This makes it straightforward to formalise a variety of higher-level CDMs and their transformations, compared with much previous work on semantic schema integration [1, 2, 4], where a specific variant of the ER model has been used as the CDM.

7 Conclusions

In this paper we have proposed a general formal framework for schema transformation based on a hypergraph data model and have defined a set of primitive transformations for this data model. We have illustrated how practical, higher-level CDMs and transformations on them can be defined in this framework by showing first how to define the binary ER schemas and primitive transformations on them that we considered in [11], and then extending the treatment to a much richer ER model supporting n-ary relations, attributes on relations, complex attributes, and generalisation hierarchies. We have defined a set of primitive transformations for this richer CDM and have shown how they can be used to express, and formalise, many of the common schema equivalences regarding n-ary relations, attributes of relations, complex attributes and generalisation hierarchies found in the literature.

Our framework is very general, and the same approach that we have used here can be adopted for formalising other CDMs and their associated primitive transformations. The fist step is to define schemas in the CDM in terms of the underlying hypergraph and additional constraints. Then primitive transformations for each construct of a schema can be defined in terms of the primitive transformations on the underlying hypergraph data model.

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 (s-d) and those which only apply for certain instances (i-d or k-b). 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 by restricting the constraints to a specific set of constraint types. A detailed theoretical treatment of our notion of schema equivalence can be found in [10], as well as a discussion of how our approach can be applied to the overall schema integration process. Informally, our approach to integrating two schemas S_1 and S_2 first applies transformations to achieve two schemas S_1' and S_2' whose common concepts [2] are either identical or compatible; these schemas can then be integrated by a simple merge of objects with the same name.

Our work has practical application in the implementation of tools for aiding schema integration. The primitive transformations can be used as a simple "programming language" for the deriving new schemas. 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/or other knowledge about the component databases (*e.g.* semantic integrity constraints). 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. We believe that our formalism is methodology-independent and could be applied to any of the methodologies proposed in literature.

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] M.V. Mannino, S.B. Navathe, and W. Effelsberg. A rule-based approach for merging generalisation hierarchies. *Information Systems*, 13(3):257–272, 1988.

[10] 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.

[11] P. McBrien and A. Poulovassilis. A formal framework for ER schema transformation. In *Proceedings of ER'97*, volume 1331 of *LNCS*, pages 408–421, 1997.

[12] 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.

[13] 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.

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

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

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

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

A Semantics of the primitive transformations

Definition 10 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 $S = \langle Nodes, Edges, Constraints\rangle$ and $S' = \langle Nodes', Edges', Constraints'\rangle$. The output extension mapping $Ext_{S',I}$ is identical to $Ext_{S,I}$ for all arguments except those explicitly defined below.

Definition 10

1. $renameNode\ \langle from, to\rangle\ \langle S, I, Ext_{S,I}\rangle \;=\; \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} S' &= S[from/to] \\ Ext_{S',I}(n[from/to]) &= Ext_{S,I}(n) \end{aligned}$$

   provided that
   (a) $to \notin Nodes$ or
   (b) $to \in Nodes$ and $Ext_{S,I}(from) = Ext_{S,I}(to)$.

2. $renameEdge\ \langle\langle from, n_1, \ldots, n_m\rangle, to\rangle\ \langle S, I, Ext_{S,I}\rangle \;=\; \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} S' &= S[from/to] \\ Ext_{S',I}(n[from/to]) &= Ext_{S,I}(n) \end{aligned}$$

   provided that
   (a) $\langle to, n_1, \ldots, n_m\rangle \notin Edges$ or
   (b) $\langle to, n_1, \ldots, n_m\rangle \in Edges$ and $Ext_{S,I}(\langle from, n_1, \ldots, n_m\rangle) = Ext_{S,I}(\langle to, n_1, \ldots, n_m\rangle)$.

3. $addConstraint\ c\ \langle S, I, Ext_{S,I}\rangle \;=\; \langle S', I, Ext_{S',I}\rangle$ such that

$$\begin{aligned} Nodes' &= Nodes \\ Edges' &= Edges \\ Constraints' &= Constraints \cup \{c\} \end{aligned}$$

provided that $VARS(c) \subseteq Nodes \cup Edges$ and $c[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$ evaluates to true, where $\{v_1, \ldots, v_n\} = VARS(c)$.

4. $delConstraint\ c\ \langle S, I, Ext_{S,I} \rangle \;=\; \langle S', I, Ext_{S',I} \rangle$ such that

$$\begin{array}{rcl} Nodes' & = & Nodes \\ Edges' & = & Edges \\ Constraints' & = & Constraints \setminus \{c\} \end{array}$$

5. $addNode\ \langle n, q \rangle\ \langle S, I, Ext_{S,I} \rangle \;=\; \langle S', I, Ext_{S',I} \rangle$ such that

$$\begin{array}{rcl} Nodes' & = & Nodes \cup \{n\} \\ Edges' & = & Edges \\ Ext_{S',I}(n) & = & q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)] \\ & & \text{where } \{v_1, \ldots, v_n\} = VARS(q) \end{array}$$

provided that $VARS(q) \subseteq Nodes \cup Edges$ and
(a) $n \notin Nodes$ or
(b) $Ext_{S,I}(n) = q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$.

6. $delNode\ n\ \langle S, I, Ext_{S,I} \rangle \;=\; \langle S', I, Ext_{S',I} \rangle$ such that

$$\begin{array}{rcl} Nodes' & = & Nodes \setminus \{n\} \\ Edges' & = & Edges \\ Ext_{S',I}(n) & = & \perp \end{array}$$

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

7. $addEdge\ \langle n_0, n_1, \ldots, n_m, q \rangle\ \langle S, I, Ext_{S,I} \rangle \;=\; \langle S', I, Ext_{S',I} \rangle$ such that

$$\begin{array}{rcl} Nodes' & = & Nodes \\ Edges' & = & Edges \cup \{\langle n_0, n_1, \ldots, n_m \rangle\} \\ Ext_{S',I}(\langle n_0, n_1, \ldots, n_m \rangle) & = & q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)] \\ & & \text{where } \{v_1, \ldots, v_n\} = VARS(q) \end{array}$$

provided that $n_1, \ldots, n_m \in Nodes \cup Edges$, $VARS(q) \subseteq Nodes \cup Edges$, and
(a) $\langle n_0, n_1, \ldots, n_m \rangle \notin Schemes$ and $q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$ satisfies the appropriate domain constraints, or
(b) $\langle n_0, n_1, \ldots, n_m \rangle \in Schemes$ and
$Ext_{S,I}(\langle n_0, n_1, \ldots, n_m \rangle) = q[v_1/Ext_{S,I}(v_1), \ldots, v_n/Ext_{S,I}(v_n)]$.

8. $delEdge\ \langle n_0, n_1, \ldots, n_m \rangle\ \langle S, I, Ext_{S,I} \rangle \;=\; \langle S', I, Ext_{S',I} \rangle$ such that

$$\begin{array}{rcl} Nodes' & = & Nodes \\ Edges' & = & Edges \setminus \{\langle n_0, n_1, \ldots, n_m \rangle\} \\ Ext_{S',I}(\langle n_0, n_1, \ldots, n_m \rangle) & = & \perp \end{array}$$

provided that $\langle n_0, n_1, \ldots, n_m \rangle$ participates in no edges and that property (ii) of Definition 2 is not violated by setting $Ext_{S',I}$ to be undefined for $\langle n_0, n_1, \ldots, n_m \rangle$.