Technical Report No.2006-522

Metamodel-independent schema & data integration: Towards joining syntax and semantics in generic model management. ⋆

Zinovy Diskin

School of Computing, Queen's University,
Kingston, Ontario, Canada
{zdiskin}@cs.queensu.ca

Abstract. The paper presents a mathematical framework, in which the main concepts of schema and data integration can be specified both semantically and syntactically in an abstract datamodel independent way. We first define *what* are schema matching and integration semantically, in terms of sets of instances and mappings between them. We also define schema matching and integration syntactically, and introduce a procedure (the *how*) for computing the integration of matched schemas according to the syntactical definition in fairly abstract terms. The main theorem of the paper states that the result of syntactical integration satisfies the semantic definition and, hence, does produce what we really need.

Viewed in a broader perspective, the framework developed in the paper integrates the syntactical and semantical sides of model management and, particularly, reveals a remarkable duality between them. The results of the paper can then be seen as an (important yet) particular case of this general duality theory.

⋆ Research supported by the Latvian Council of Science (Grant #05.1526), OCE Centre for Communications and Information Technology and IBM CAS Ottawa.

1 Introduction: motivating discussion, relation to other works and an outline of the approach

1.1 What this paper is about

Schema and data integration is a classical problem for the database theory, with an enormous literature devoted to it. The majority of works in the field is focused on the procedures of working with syntactical objects (schemas and their transformations) while their semantics is left implicit. It is usually hoped that everything is well on the semantic side too, but without explicit models of the semantic aspects, this hope remains a subject of belief rather than verification.[1]

Semantic issues become especially actual, if not crucial, in the heterogenous situation, when data and schemas in essentially different datamodels need to be integrated. Lately, Semantic web, data warehousing and enterprize integration systems dramatically increased the interest in heterogeneous data integration and exchange, and in their theoretical support as well. Not surprisingly, it has caused the turn to semantics in the field [10, 3, 18, 8, 9, 13].

However, there is a hidden obstacle in working with semantics of schema integration (often also called model merge), and other model management (MMt) operations. Consider two data schemas, say, relational, S and T, with the sets of instances $\boldsymbol{inst}(S)$ and $\boldsymbol{inst}(T)$. Suppose that these two sets are countable and hence there is a bijective mapping b between them. We can say that schemas S and T are equally *instance capable*. However, bijection b provides no meaningful information about the relative *information capacities* of the schemas because, as noted in [15], "... schema instances with no intuitive relationship between them are allowed to be associated via the mapping".

The situation radically changes when we first take a mapping between schemas, $p\colon S \to T$, that we consider as an interpretation or simulation of T-elements by S-elements. Then any instance of T can be also considered as an instance of S by "composition" with p (this construction will be precisely defined in the paper). The result is that we have a mapping $[\![\, p \,]\!]\colon \boldsymbol{inst}(T) \to \boldsymbol{inst}(S)$. Now, if it is known that $[\![\, p \,]\!]$ is a bijection, it gives us much more information about relative information capacity of schemas S and T than an "accidental" bijection b above. Still, however, we cannot assert that S and T are equally information capable because the inverse mapping, $([\![\, p \,]\!])^{-1}\colon \boldsymbol{inst}(S) \to \boldsymbol{inst}(T)$, is "semi-accidental". If this inverse mapping would be the $[\![-]\!]$ image of some interpretation between the schemas, that is, $([\![\, p \,]\!])^{-1} = [\![\, r \,]\!]$ for some mapping $r\colon T \to S$, then we would have a meaningful (interpretable) isomorphism between the instance sets of the schemas and could say that they have equal information capacities.

Is the class $E = \{f\colon \boldsymbol{inst}(T) \to \boldsymbol{inst}(S) \mid f = [\![\, p \,]\!] \text{ for some } p\colon S \to T\}$ of expressible mappings is big enough to cover all meaningful cases? The answer is negative since there are many interesting and practically useful interpretations

[1] This consideration is of course well understood in the community, and the focus on syntax is caused by the infamous difficulties of building semantic models, not because their value is underestimated.

between schemas not covered by the class of plain interpretations p as above. Namely, we can consider an interpretation that assigns to elements of schema S elements computed by queries over schema T rather than elements of T as such. In fact, such interpretations are well known under the name of *views* (to schema T), and we will denote them by, say, double arrows $v\colon S \Rightarrow_{\mathbf{Q}} T$ with $\mathbf{Q}$ referring to the query language in question. In this way we come to a class of expressible mappings $E_{\mathbf{Q}} = \{f\colon \boldsymbol{inst}(T) \to \boldsymbol{inst}(S) \mid f = [\![\, v \,]\!] \text{ for some } v\colon S \Rightarrow_{\mathbf{Q}} T\}$, which is much broader than E if the query language $\mathbf{Q}$ is expressive enough.

In fact, many recent works in data integration and exchange do consider the class $E_{\mathbf{Q}}$ but do not infer it from the class of syntactically defined view mappings. The latter are considered either semi-formally like in [10] or formally but within the relational data model as in [9], or the syntactical side is at all disregarded as in [13]. Thus, the case of *generic* (datamodel independent) formalism, where both the syntactical and semantic sides of the issue are modeled, is not covered in the literature. Note that this problem was explicitly stated in [3] as one of the most important issues in the generic MMt, and it is the problem where the present paper is intended to contribute.

1.2 Why category theory.

A major obstacle in approaching the problem is in a generic formal model of the notion of view mapping. Indeed, it is clear from the discussion above that it has to involve a generic notion of query language. There are many particular query languages, and many of them are very well understood formally (with the relational algebra RA as a notable example). However, it is not easy to formulate what a query language is in general. In other words, what is that hypothetic abstract formal pattern/definition of the notion, which would include as particular special cases such languages as, say, RA, SQL, XQuery and, say, graph-based languages for querying graph-based data schemas like UML or ER diagrams? As a query language is a (suitably defined) algebra, the problem can be reformulated in pure mathematical terms as "what is an algebra in general"? Although algebra is as ancient as mathematics, and plenty of useful and perfectly formal algebraic notions were developed, figuring out what algebra is in general and building the corresponding formal framework turned out to be a highly non-trivial problem. It was solved relatively recently (in the sixties-seventies) in the mathematical Category Theory (CT), see [12] for a detailed presentation of the ideas and [2] for a compendium of useful results. This formal framework is usually called *categorical algebra* (CA).

An idea directly suggested by CA and heavily employed in this paper is to consider a view "mapping" $v\colon S \Rightarrow_{\mathbf{Q}} T$ as an ordinary (elements to elements) mapping between schemas, $v\colon S \to \mathsf{der}_q T$, where $\mathsf{der}_q T$ is a suitable augmentation of schema T with derived elements computed by some query q against schema T. Roughly, schema $\mathsf{der}_q T$ can be thought of as the union of schema T and the schema of the query q. For example, think of adding to relational schema T new tables defined by a relational query q. In fact, a similar idea was used in [4, 3], where query expressions were assigned directly to the mappings

between schemas rather than to the target schemas. An essential benefit of the $\mathsf{der}_q T$-approach is that view "mappings" become ordinary mappings that can manipulated in the ordinary way. (Note also that schema integration is really saturated with mappings of the form $v\colon S \to \mathsf{der}_q T$, since it is a typical situation of schema overlapping when elements of one schema correspond to elements, which are not immediately presented in another schema but can be derived in it by posing suitable queries, see [5] for a detailed discussion).

On the other hand, the original MMt-idea of attributing query expressions to mappings rather than schemas can also be useful, because it allows us to consider two *different views* to the *same schema* T as two *different arrows* to the *same target node*. A consistent realization of this idea is also possible, but it requires us to respect some delicate issues in working with arrows, and here categorical algebra provides guidance that is difficult to overestimate. Note that developing a convenient machinery for working with arrows coupled with query expressions was also listed in [3] as an important MMt problem.

To summarize, a theoretical support for current schema and data integration and exchange problems needs a sufficiently abstract framework, where both the syntactical and the semantic sides of the issue would be consistently modeled in a generic way. The main difficulty in building such a framework is in a generic model of the syntactical side (query languages and views). Though a few crucial ideas have already appeared in the literature and are known to the community, their consistent, accurate and well-formalized presentation seems to be absent so far.[2] On the other hand, many necessary concepts and a respective machinery are already developed in categorical algebra (CA). What should thus be fruitful for the problems in question (and for generic MMt on a whole as well) is to adapt CA-constructs to the field and explore how useful their application might be. This is a broad agenda (still to be justified), and the present paper makes an initial step; an earlier and less systematic attempt can be found in [6].

1.3 About the presentation and results

Category theory (CT) is notorious even in pure mathematics for an excessively abstract nature of its constructs[3]; this is the payment for the capability to formalize such notions as "algebra in general" or "xxx in general". Any author willing to present categorical machinery for a non-categorical audience is forced to sail between the Scylla of being formally perfect yet overloaded with excessive technical details, and the Charybdis of being intuitively understandable but formally approximate with many important formal details omitted. The present author has chosen a path closer to the latter monster (but hopefully not in a dangerous proximity to be entirely eaten by "warm and fuzzy" imprecision :-). Our route in the paper will be as follows.

[2] The paper [1] does present a formal framework for encompassing both syntax and semantics of model management but the treatment is a way too abstract; particularly, derived information/querying is not considered at all.

[3] called *abstract nonsense* during CT's early days

We will begin with a few simple notational conventions and semantic definitions in sections 2 and 3, including, particularly, a revised definition of information capacity (IC-)equivalence between schemas introduced in [15]. We need it for a correct formulation of schema integration results because we show that the integrated schema is defined up to IC-equivalence. Generic considerations of semantics are not too hard, and the definitions in these sections are almost completely formal. The main body of the paper is in sections 4 and 5 where a simple example of relational schema integration is discussed. However, the way we will discuss this particular example is, in fact, categorical and hence ready to be made generic (datamodel independent). Namely, we will formulate our particular results in terms of graphs, whose nodes denote schemas and arrows are mappings between them. Other necessary constructions we encounter in the example are also formulated in the graph-based terms of diagram operations and predicates, and mappings between graphs (functors). Though for this particular example such a generality may seem somewhat excessive (yet all constructs will appear quite naturally and with a clear relational interpretation), it is justified by the readiness of the framework for a truly generic reformulation. Indeed, all that we need to do for this end is (a) to make our graph-based arrangement complete and consistent (thus entirely eliminating all the peculiarities of the relational datamodel) and (b) to interpret everything in abstract terms by considering nodes and arrows as schemas and their mappings in an arbitrary datamodel. "Arbitrary" here means arbitrary in a very wide class including the relational, XML, graph-based (ER, UML and the like) and many other datamodels.[4] Of course, a complete realization of the task (a) needs a lot of formal details, and an accurate formal description is omitted due to space limitations. Some outline of the formal picture can be found in [6], a full description will appear elsewhere taking the present paper as a motivation.

Among results presented in the paper are

(i) a quite general pattern for specifying correspondences between schemas to be integrated (in both syntax and semantics);

(ii) independent semantic and syntactical definitions of schema integration;

(iii) a theorem stating that the semantic and syntactic definitions (ii) have the same extension and, hence, any algorithm computing the merge according to the syntactic definition will produce what we really need semantically;

(v) a general framework showing a remarkable duality between the syntactical and semantic sides of schema integration.

2 Preliminary definitions and notational conventions

In this section we outline a framework of necessary definitions in a semi-formal style. Making them perfectly formal may itself be non-trivial, part of this work will be done later. Syntactically, a schema (relational, ER-diagram, DTD, UML

[4] Category theory allows us to formally specify the class of datamodels to which specifications presented in the paper are applicable, and this class turns out to be really broad and encompassing all models listed above and more,see [**?**] for details.

class diagram) is a structured finite set of *elements* or *items*. Semantically, the fundamental assumption is that a non-empty consistent schema S has a non-empty set of *instances*, $\boldsymbol{inst}(S)$. Instances will be denoted by small bold letters, maybe, indexed, like, e.g., $\boldsymbol{i}_1$ or $\boldsymbol{k}$. Many our formulas will be much more readable with another notation for the set $\boldsymbol{inst}(S)$, namely, $[\![S]\!]$. Thus, expressions $\boldsymbol{inst}(S)$ and $[\![S]\!]$ are synonyms. A reasonable use of synonymous notation can greatly facilitate readability and hopefully does not create real problems.

Given two schemas S, T, expression $[\![S]\!] \cong [\![T]\!]$ means that there is some uniquely defined/canonic/meaningful isomorphism between the two sets rather than an "accidental" bijection. Particularly, if two schemas are isomorphic via some isomorphism $i\colon S \xrightarrow{\cong} T$ (syntactical details are irrelevant), the sets of instances are canonically isomorphic by the isomorphisms $[\![i]\!]\colon [\![T]\!] \xrightarrow{\cong} [\![S]\!]$ and $\big[\!\big[i^{-1} \big]\!\big]\colon [\![S]\!] \xrightarrow{\cong} [\![T]\!]$, and we may write $[\![S]\!] \cong [\![T]\!]$. We will also write $i : S \cong T$ or just $S \cong T$ to denote isomorphism of schemas.

A query to schema S is an expression/term q in some query language, which is formed by S-items and operation symbols of the language; syntactical details of these expressions are again irrelevant. What matters is that q has its own schema S_q and that it is a query against S, we write this as $q\colon S_q \rightsquigarrow S$. Semantically, a query is a mapping $[\![q]\!]\colon [\![S]\!] \to \big[\!\big[S_q \big]\!\big]$ (note the reversal of the arrow), where a *mapping* means a single-valued totally-defined mapping (function). Following a common practice, we will often write just q instead of $[\![q]\!]$. We write $q(\boldsymbol{i})$ or $\boldsymbol{i}.q$ or $\boldsymbol{i}^q$ for the result of applying the query to instance $\boldsymbol{i}$ (of course, the same notation will be used within the same formula).

Note that every query q gives rise to a mapping $[\![q]\!]$ between instances but the converse is not true. Given two schemas S, T, a mapping $f\colon [\![S]\!] \to [\![T]\!]$ is called *expressible* if there is a query $q\colon S_q \rightsquigarrow S$ whose schema is isomorphic to T, $i : T \cong S_q$, and such that $f = [\![q]\!] \Rrightarrow [\![i]\!]$, where $\Rrightarrow$ denotes the operation of mapping composition.

The difference between ordinary and expressible mappings between instance sets is a fundamental issue for the subject. Particularly, it motivates the following reformulation of the notion of schema equivalence w.r.t. their information capacities discussed in [15]. Two schemas S_1, S_2 are called *equivalent w.r.t. their information capacities* or *IC-equivalent*, $S_1 \equiv S_2$, if there are two queries $q_1\colon S_{q_1} \rightsquigarrow S_1$ and $q_2\colon S_{q_2} \rightsquigarrow S_2$ such that
(i) $i_1 : S_2 \cong S_{q_1}$, $[\![i_1]\!]\colon [\![S_{q_1}]\!] \xrightarrow{\cong} [\![S_2]\!]$;
(ii) $i_2 : S_1 \cong S_{q_2}$, $[\![i_2]\!]\colon [\![S_{q_2}]\!] \xrightarrow{\cong} [\![S_1]\!]$;
(iii) and mappings $([\![q_1]\!] \Rrightarrow [\![i_1]\!])\colon [\![S_1]\!] \to [\![S_2]\!]$ and $([\![q_2]\!] \Rrightarrow [\![i_2]\!])\colon [\![S_2]\!] \to [\![S_1]\!]$ set an isomorphism between $[\![S_2]\!]$ and $[\![S_1]\!]$.

A *graph* means a directed graph with the possibility of several arrows between the same two nodes (multigraph). We will often deal with the following two configurations of nodes and arrows: *spans*, which are pairs of arrows with a common source, and *cospans*, which are pairs of arrows with a common target.

If S is a data schema (relational, ER-diagram, DTD, UML class diagram) and *name* is the name of its element (table, column, node), we write *name*@S

to distinguish it from equally named elements of other schemas. If schemas or mappings are computed by some formal operation/automatic procedure, we call them *derived*. In our diagrams, derived objects are depicted with dashed blue (grey in black-white printing) lines.

3 Instance-based semantics of schema integration

When we consider instances of different schemas, it is useful to explicate their synchronization/simultaneity relationship. We can do that by introducing some hypothetical "world schema" W such that all instances of any ordinary database schema S can be considered as the corresponding projections of the "world instances". In other words, for any schema S there is a mapping $w_S\colon \boldsymbol{inst}(W) \to \boldsymbol{inst}(S)$. This world schema will not, of course, occur in our syntactical procedures but is useful for formulating some important semantic conditions.

Now consider two simple relational schemas in the top part of Fig. 1. The schema S_1 consists of two tables, Orders and Hardware, each having three columns whose names are written under the table names. The second schema consists of one three-column table Customer. Suppose that somehow we know that the set of customers referred to by schema S_2 is exactly the set of those customers for schema S_1, who ordered products of vendor "HP". This latter set can be computed by an evident query q_1 specified in schema $\mathsf{der}_{q_1} S_1$ in Fig. 1. In addition, for these two sets of customers, the columns $custName@S_1$ and $name@S_2$ must contain the same values.[5]

Semantically, these conditions mean that if $\boldsymbol{k}$ is a world instance and $\boldsymbol{i}_j = \boldsymbol{k}.w_{S_j}$ $(j = 1, 2)$ are the corresponding (synchronized) local instances, then the equality $\boldsymbol{i}_1.q_1 = \boldsymbol{i}_2.q_2$ holds. Here $q_j\colon S_0 \rightsquigarrow S_j$, $j = 1, 2$ are the queries described in the conditions and S_0 is their common schema (see Fig. 1). In other words, the conditions state the commutativity of the triangle diagram (1) in Fig. 2

3.1 Definition: problem of schema matching, semantically. Given two schemas S_1,S_2 called local, the *schema match problem* is to find a pair of queries $q_j\colon S_0 \rightsquigarrow S_j$ $(j = 1, 2)$ with a common schema $S_0 = S_{q_1} = S_{q_2}$ such that diagram (1) in Fig. 2 commutes. In such a case, the triple (cospan) $\mathcal{Q} = (S_0, q_1, q_2)$ of type $\Big(\llbracket\, S_1 \,\rrbracket \xrightarrow{q_1} \llbracket\, S_0 \,\rrbracket \xleftarrow{q_2} \llbracket\, S_2 \,\rrbracket\Big)$ is called a *semantic match* for the pair (S_1,S_2).

As the world instances are only imaginary entities, in practice we cannot formally check the commutativity condition. Nor can we formally check whether a cospan $\mathcal{Q}$ captures all or just part of the overlapping between the local schemas. Schema matching is a heuristic activity providing the input information for the integration algorithm but not its formal part. Of course, it does not exclude a possibility of building special schema matching algorithms based on inductive learning and other similar techniques developed in AI (see [11, 17] for some results).

[5] How can we obtain such a knowledge is, of course, a special issue considered in the discipline of schema matching in-between databases, AI and pragmatics, see [17] for a survey and [7] for a brief discussion.

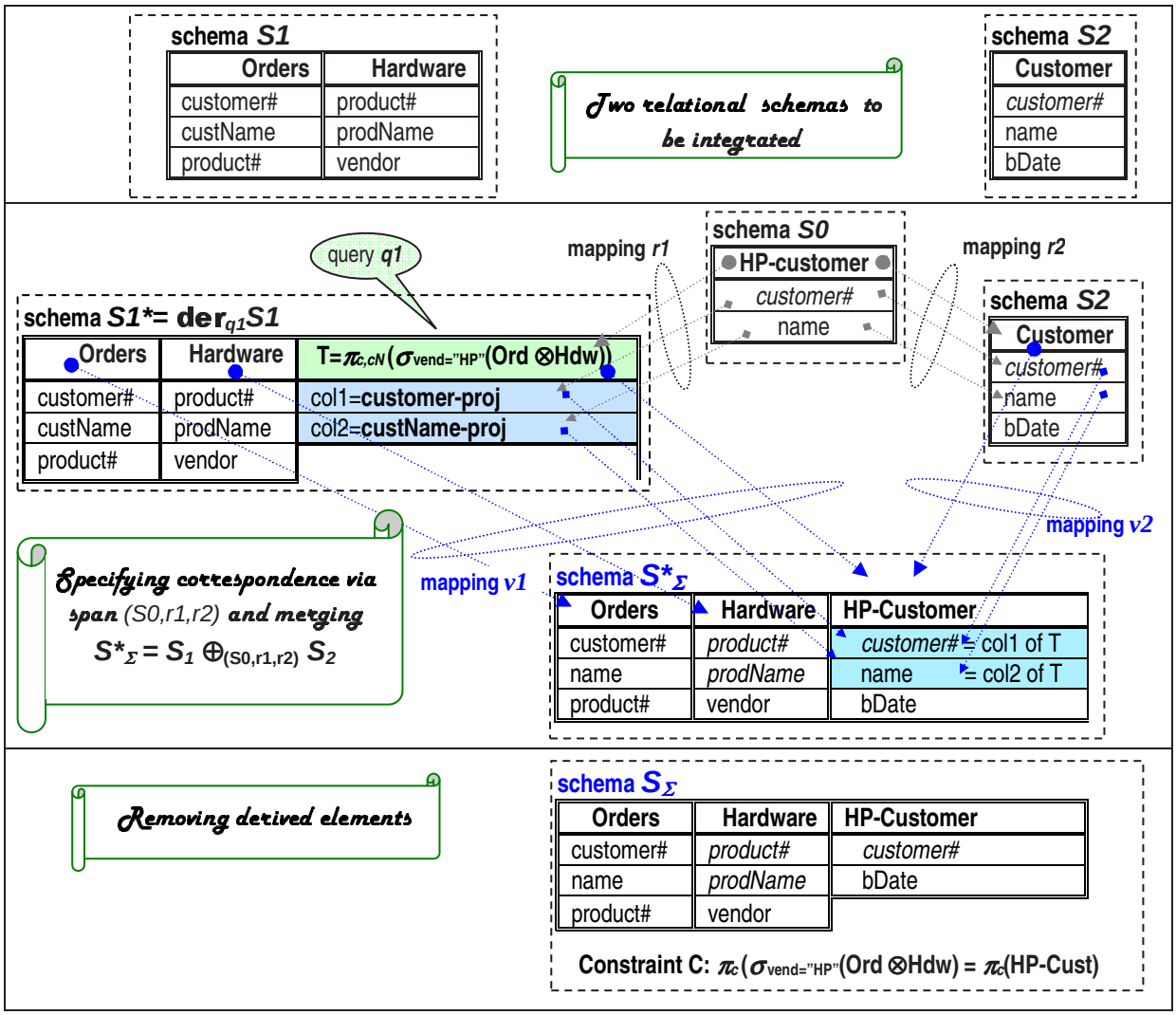

Fig. 1. Example of schema integration. [Derived elements are shaded (and blue in color print) . In long expressions, names of tables and columns are abbreviated by few letters.]

3.2 Definition: problem of schema integration, semantically. Given a cospan $\mathcal{Q}$ of matched schemas as above, find a span $\mathcal{G} = (G, g_1, g_2)$ of type $\Big(\llbracket S_1 \rrbracket \xleftarrow{g_1} \llbracket G \rrbracket \xrightarrow{g_2} \llbracket S_2 \rrbracket\Big)$, where G is a schema called *global* and $g_j : \llbracket G \rrbracket \to \llbracket S_j \rrbracket$, $(j = 1, 2)$ are two queries to it from the local schemas, $g_j : S_j \rightsquigarrow G$, such that triangle diagram (2) in Fig. 2 commutes.

Evidently, there may be many global spans satisfying the condition above. We say that a global span $(G!, g!_1, g!_2)$ is *exact* or *(exactly) integrates* the local schemas modulo their match $\mathcal{Q}$, if it possesses the following *universal* property in the totality of all global spans: for any global span (G, g_1, g_2), there is a uniquely defined query mapping $! : \llbracket G \rrbracket \to \llbracket G! \rrbracket$ making all diagrams in Fig. 2 commutative.

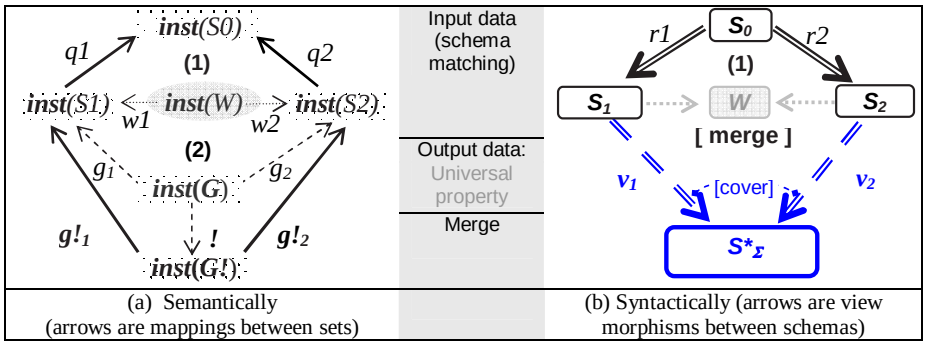

Fig. 2. Schema integration

Note a principle distinction of the schema integration problem from that of schema matching: in the former, we can formulate commutativity and universality conditions and check them without the imaginary world instances. In fact, these conditions define a binary predicate $\mathit{Merge}(\mathcal{X}, \mathcal{Y})$ over the universe of spans of the specified type, and a span $\mathcal{G} = (G, g_1, g_2)$ integrates the local schemas matched by a cospan $\mathcal{Q}$ iff the sentence $\mathit{Merge}(\mathcal{Q}, \mathcal{G})$ is true, we write $\models \mathit{Merge}(\mathcal{Q}, \mathcal{G})$.

The universal property allows us to prove that the integrated schema, if it exists, is uniquely determined up to its information capacity.

3.3 Proposition. If $\mathcal{G}! = (G!, g!_1, g!_2)$ and $\mathcal{G}!! = (G!!, g!!_1, g!!_2)$ are two integrations of a cospan $\mathcal{Q}$, that is, $\models \mathit{Merge}(\mathcal{Q}, \mathcal{G}!)$ and $\models \mathit{Merge}(\mathcal{Q}, \mathcal{G}!!)$, then $G!$ and $G!!$ are IC-equivalent, $G! \equiv G!!$.

Proof. Universality of $G!$ provides the existence of a unique query $! : \llbracket G!! \rrbracket \to \llbracket G! \rrbracket$ and universality of $G!!$ provides $!! : \llbracket G! \rrbracket \to \llbracket G!! \rrbracket$. Commutativity of the respective diagram means that these query mappings are mutually inverse.

Note that definition 3.2 is declarative/denotational rather than operational: it does not give an algorithm of actual building the integrated schema nor does it give any evidence that such a schema actually exists.

Now we are going to put our abstract semantic definitions on hold for a while and consider the example in Fig. 1 in quite concrete syntactical terms.

4 Example of schema integration, syntactically

4.1 Match and merge

Our first step is to specify the match/correspondence between schemas syntactically rather than semantically. To this end, we augment schema S_1 with a derived table T with columns col1, col2 computed as prescribed by query q_1 specified in Fig. 1 (the derived elements are shaded). We denote the augmented schema by $\mathsf{der}_{q_1} S_1$ or ${S_1}^*$. We then introduce a new schema S_0 with the only table HP-Customer and show how this table is represented in the local schemas. This work is done by schema mappings $r_j \colon S_0 \to {S_j}^* = \mathsf{der}_{q_j} S_j$ $(j = 1, 2)$ targeted into local schemas augmented with derived elements (for our particular example, ${S_2}^* = S_2$). Following category theory, we will call such mappings *Kleisly morphisms*. Thus, now the correspondence information is specified by a span $\mathcal{R} = (S_0, r_1, r_2)$ of type $\left(\mathsf{der}_{q_1} S_1 \xleftarrow{r_1} S_0 \xrightarrow{r_2} \mathsf{der}_{q_2} S_2\right)$. In contrast to semantic match $\mathcal{Q}$ (which is a cospan), arrows in the span $\mathcal{R}$ are purely syntactic entities: they map schemas as sets of elements rather than sets of instances.

The span $\mathcal{R}$ can be thought of as a set of equalities between local schema elements like $\mathit{col1}@{S_1}^* = \mathit{customer\#}@S_2$ and $\mathit{col2}@{S_1}^* = \mathit{name}@S_2$, which should be used to eliminate unnecessary duplication of schema elements in their merge. A natural next step is to merge the augmented local schemas disjointedly and then factorize the merge modulo the equalities. The latter means that we glue together those elements which are images of the same element in schema S_0. Roughly speaking, it will make the resulting schema S^*_Σ a disjoint union of the three components:

$$S^*_\Sigma \cong ({S_1}^* \setminus S_0) \uplus S_0 \uplus ({S_2}^* \setminus S_0). \tag{1}$$

Speaking more accurately, we need to consider the corresponding mapping between schemas, and consider the result of integration to be a cospan $\mathcal{S}^* = (S^*_\Sigma, v_1, v_2)$ specified in Fig. 1, see also Fig. 3 (ii).

To finish integration, we need to remove from S^*_Σ derived (shaded) items. Unfortunately, in general this process is not trivial since removal derived items may violate some structural requirements to schemas. For instance, in our example, removing the derived column *cutomer#* will leave the table HP-Customer without its primary key. Hence, we forced to keep it in the schema but then add to it a corresponding integrity constraint (see Constraint C in Fig. 1).

Thus, an important final part of the integration procedure should be a special procedure of schema *normalization*. The latter is aimed at finding some schema

S_N with minimal redundancy but equivalent to the merge schema in the following sense: each element of the merge schema $S_\Sigma{}^*$ can be derived from S_N (no information is lost) and conversely, each element of S_N is derivable from $S_\Sigma{}^*$ (nothing extra is acquired). It is easy to see that our schema S_Σ can be further normalized but a detailed discussion goes beyond the goals of this paper.

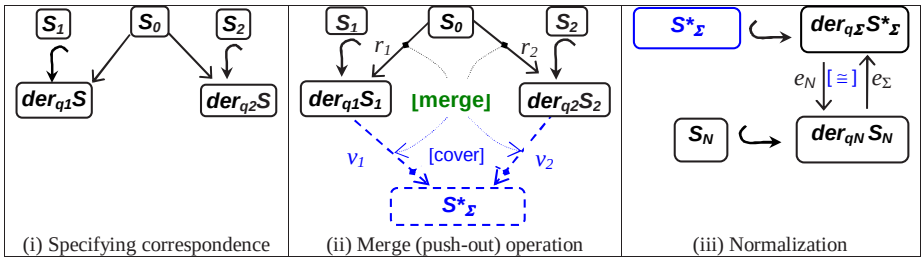

Fig. 3. Schema integration syntactically

4.2 Towards abstraction: Views as Kleisly morphisms

Consider Fig. 3 where we make a step to abstraction from peculiarities of our relational example and present the syntactical procedures above in an abstract graph-based notation. Nodes denote schemas considered as sets of elements and arrows are mappings between them; hooked arrows denote set inclusions. Label merge denotes the operation of disjointed union of the two schemas followed by factorization according to the "equalities" specified by span $\mathcal{R} = (S_0, r_1, r_2)$. The result is a merge cospan $\mathcal{S}_\Sigma = (S^*_\Sigma, v_1, v_2)$. In the linear notation we can write this as $\mathcal{S}_\Sigma = \mathsf{merge}(\mathcal{R})$.

We see that arrows of the form $v\colon S_v \to \mathsf{der}_q S$ play a fundamental role in schema integration (it was also noted in [5]). In fact, such arrows formally model the well-known notion of *view* (with S_v being the view schema). This interpretation suggests the following useful rearrangement of our notation.

In the view specification as above, we can attribute the reference to query q to the very arrow rather than to its target and rewrite the view expression as $v_q\colon S_v \Rightarrow S$, where the double arrow $\Rightarrow$ denotes a mapping into $\mathsf{der}_q S$ rather than S.[6] Particularly, two different views to schema S with the same schema $S_{v1} = S_{v2}$ will be two different arrows with the same target. We will call these double arrows *view mappings* or, following the terminology adopted in category theory, *Kleisly mappings/morphisms*. Note that queries become trivial view/Kleisly arrows for which $\mathsf{der}_q S = S \uplus S_q$ and the view mapping v is nothing but the canonical embedding of S_q into $S \uplus S_q$. Thus, our query arrows $\rightsquigarrow$ are just special view

[6] To be accurate, we need to attach to the arrows a reference to the query language. Since our query language is fixed, we omit the reference.

arrows. With view mappings, the procedure of schema integration is specified in Fig. 2(b).

5 Syntax and semantics together: united yet dual

Diagrams in Fig. 2 present our work with the relational example in a compact and observable way way. We at once note a remarkable similarity between the semantic and syntactic sides of the issue. However, so far these sides are not related. The diagram (a) presents a definition, that is, a predicate $Merge(\mathcal{X}, \mathcal{Y})$ over spans in the universe of instance sets and view/query mappings. However, it does not give any procedural clue to how to find the integrated spans nor proves their existence. In contrast, diagram (b) is procedural: it refers to an operation/procedure merge that, given an input span $\mathcal{R} = (S_0, r_1, r_2)$, computes its merge, a cospan $\mathcal{S}_\Sigma = (S^*_\Sigma, v_1, v_2)$. We will briefly write it as $\mathcal{S}_\Sigma = \mathsf{merge}(\mathcal{R})$.

The major question is how the syntactical and semantic sides of schema integration are related. Particularly, what is semantic meaning of our merge cospan $\mathcal{S}_\Sigma$ built syntactically? Can we extract from it a global span semantically integrating schemas as defined in definition 3.2? These are the questions to be answered in this section.

5.1 Instances as mappings

In our syntactical section 4 we worked with mappings between relational schemas. These mappings send table names to table names and column names to column names preserving the columns' domains and the containment relation between tables and columns. The key to specifying syntax-semantics relationships is an observation that schema instances can be also considered as schema mappings. Let our relational schemas be created over some set $\mathcal{D}$ of basic domains/SQL types. Consider a huge relational schema U, whose "table names" are all possible tables populated with data values from $\mathcal{D}$, that is, all relations over $\mathcal{D}$. The "column names" are the columns of data values, that is, the projection mappings of the relations. Then instances $\boldsymbol{i}$ of schema S can be considered as mappings $\boldsymbol{i}\colon S \to U$ into U. [7]

Given a query q against S, any S-instance can extended to an instance $\mathsf{der}_q\boldsymbol{i}\colon \mathsf{der}_q S \to U$ in a unique way. In more detail, we first extent $\boldsymbol{i}$ to an instance $\mathsf{der}_q\boldsymbol{i}\colon \mathsf{der}_q S \to \mathsf{der}_q U$ homomorphically by substitution. Then we use a special nature of schema U: since U's elements have actual contents (extension) over which queries really operate, any query $q(u_1...u_n)$ over U can be evaluated to a real element (*ie*, a relation) in U; it gives rise to a mapping $\alpha_q\colon \mathsf{der}_q U \to U$. Finally, composition of this mapping with $\mathsf{der}_q\boldsymbol{i}$ gives a mapping from $\mathsf{der}_q S$ to U, which by an abuse of notation we also denote by $\mathsf{der}_q\boldsymbol{i}$. Conversely, any instance $\boldsymbol{i}\colon \mathsf{der}_q S \to U$ can be reduced to its projection $\boldsymbol{i}\rceil_S\colon S \to U$. In this way we set a canonical isomorphism between sets $\boldsymbol{inst}(S)$ and $\boldsymbol{inst}(\mathsf{der}_q S)$ and show that schemas S and $\mathsf{der}_q S$ are IC-equivalent (as it should be).

[7] Add a few words about integrity constraints.

Now let $v\colon S_v \to \mathsf{der}_q S$ be a view (Kleisly mapping) to schema S, and $\boldsymbol{i}\colon S \to U$ be an S-instance. By extending $\boldsymbol{i}$ to $\mathsf{der}_q\boldsymbol{i}\colon \mathsf{der}_q S \to U$ and then composing it with v, we obtain an S_v-instance $v \mathbin{⊳} \mathsf{der}_q\boldsymbol{i}\colon S_v \to U$. In this way any view mapping v between schemas gives rise to a mapping between the corresponding instance sets going in the opposite direction. We will denote this mapping by $[\![\, v \,]\!]$, that is, $[\![\, v \,]\!](\boldsymbol{i}) \stackrel{\text{def}}{=} v \mathbin{⊳} \mathsf{der}_q\boldsymbol{i}$.

Thus, any schema S is assigned with its sets of instances $[\![\, S \,]\!]$, and any view mapping $v\colon S_v \to \mathsf{der}_q S$ gives rise to the mapping $[\![\, v \,]\!]\colon [\![\, S \,]\!] \to [\![\, S_v \,]\!]$ (note the reversal of arrows). Then any configuration/diagram $\mathcal{C}$ in the universe of schemas and view mappings between them is mapped to a similar yet dual (with reversed arrows) configuration $[\![\, \mathcal{C} \,]\!]$ in the universe of sets (of instances) and functions between them. Particularly, a correspondence span $\mathcal{R} = (S_0, r_1, r_2)$ between schemas S_1,S_2 is mapped to a matching cospan $[\![\, \mathcal{R} \,]\!] = ([\![\, S_0 \,]\!], [\![\, r_1 \,]\!], [\![\, r_2 \,]\!])$. Table 1 summarizes these observations.

Table 1. Syntax-vs-semantics duality, I

Syntax	Schema, S	View mapping, $v\colon S_v \to \mathsf{der}_q S$ or $v\colon S_v \Rightarrow S$	Correspondence span, $S_1 \xleftarrow{r_1} S_0 \xrightarrow{r_2} S_2$
Semantics	Instance set, $[\![\, S \,]\!]$	Query mapping, $[\![\, v \,]\!]\colon [\![\, S \,]\!] \to [\![\, S_v \,]\!]$	Match cospan, $[\![\, S_1 \,]\!] \xrightarrow{[\![\, r_1 \,]\!]} [\![\, S_0 \,]\!] \xleftarrow{[\![\, r_2 \,]\!]} [\![\, S_2 \,]\!]$

The following theorem adds to the table the column about schema integration.

5.1 Theorem: syntax-vs-semantics duality, II. Let S_1,S_2 be two schemas with a correspondence span

$\mathcal{R} = (S_0, r_1\colon S_0 \Rightarrow S_1, r_2\colon S_0 \Rightarrow S_2)$ between them, and cospan

$\mathcal{S}_\Sigma = (S_\Sigma, v_1\colon S_1 \Rightarrow S_\Sigma, v_2\colon S_2 \Rightarrow S_\Sigma)$ is the result of syntactical integration (in the universe of schemas and view mappings), $\mathcal{S}_\Sigma = \mathsf{merge}(\mathcal{R})$. Then the statement $\models \mathit{Merge}([\![\, \mathcal{R} \,]\!], [\![\, \mathcal{S}_\Sigma \,]\!])$ is true. In other words, our syntactical integration procedure produces a semantically required result.

Proof. Let $\mathcal{S}_\Sigma = \left(S_1 \xrightarrow{v_1} S_\Sigma \xleftarrow{v_2} S_2\right)$ be the merge cospan of some correspondence span $\mathcal{R} = \left(S_1 \xleftarrow{r_1} S_0 \xrightarrow{r_2} S_2\right)$ as defined in sect. 4, $\mathcal{S}_\Sigma = \mathsf{merge}(\mathcal{R})$. Then the following two lemmas hold.

5.2 Lemma: a universal property of syntactic schema merge. For any other cospan $\mathcal{S}_\Sigma{}' = \left(S_1 \xrightarrow{v_1'} S_\Sigma{}' \xleftarrow{v_2'} S_2\right)$ making a commutative square with $\mathcal{R}$, there is a uniquely defined view mapping $!\colon S_\Sigma \Rightarrow S_\Sigma{}'$ making all the diagrams involved commutative. In other words, the merge cospan is minimal among all cospans making the diamond $\mathcal{R}\Diamond\mathcal{S}'$ commutative.

Proof. Consider the structure of merge schema in the presentation (1) on p.10.

5.3 Lemma: an element-wise specification of $[\![\,\mathcal{S}_\Sigma\,]\!]$. Let $\mathsf{PB}([\![\,\mathcal{R}\,]\!])$ denotes the following set specified element-wise (*ie*, via its elements):

$$\mathsf{PB}([\![\,\mathcal{R}\,]\!]) \stackrel{\text{def}}{=} \{(\boldsymbol{i}_1, \boldsymbol{i}_2) \in [\![\,S_1\,]\!] \times [\![\,S_2\,]\!] \mid [\![\,r_1\,]\!](\boldsymbol{i}_1) = [\![\,r_2\,]\!](\boldsymbol{i}_2)\}\,.$$

Then there is a canonically defined isomorphism $[\![\,\mathcal{S}_\Sigma\,]\!] \cong \mathsf{PB}([\![\,\mathcal{R}\,]\!])$.

Proof. By definition of functor $[\![\,.\,]\!]$,

$$\mathsf{PB}([\![\,\mathcal{R}\,]\!]) = \{(\boldsymbol{i}_1, \boldsymbol{i}_2) \in [\![\,S_1\,]\!] \times [\![\,S_2\,]\!] \mid r_1 \Rrightarrow \boldsymbol{i}_1 = r_2 \Rrightarrow \boldsymbol{i}_2\}$$

Then for any pair $(\boldsymbol{i}_1, \boldsymbol{i}_2) \in \mathsf{PB}([\![\,\mathcal{R}\,]\!])$ there exists a unique mapping $!\colon S_\Sigma \to U$ by the universal property of S_Σ (Lemma 5.2). It gives rise to a mapping $f1\colon \mathsf{PB}([\![\,\mathcal{R}\,]\!]) \to [\![\,S_\Sigma\,]\!]$. Conversely, if $k\colon S_\Sigma \to U$ is an instance of S_Σ, it gives rise to the pair of instances $v_j \Rrightarrow k \in [\![\,S_j\,]\!]$, $j = 1, 2$. It makes a mapping $f2\colon [\![\,S_\Sigma\,]\!] \to \mathsf{PB}([\![\,\mathcal{R}\,]\!])$. Owing to universality, these mappings are mutually inverse and, hence, $[\![\,S_\Sigma\,]\!]$ and $\mathsf{PB}([\![\,\mathcal{R}\,]\!])$ are canonically isomorphic, $[\![\,S_\Sigma\,]\!] \cong \mathsf{PB}([\![\,\mathcal{R}\,]\!])$.

Now let span $\mathcal{G}! = \left([\![\,S_1\,]\!] \xleftarrow{g!_1} [\![\,G!\,]\!] \xrightarrow{g!_2} [\![\,S_2\,]\!]\right)$ integrates cospan $[\![\,\mathcal{R}\,]\!]$ as defined in definition 3.2, that is, $\models \mathit{Merge}([\![\,\mathcal{R}\,]\!], \mathcal{G}!)$. The universality property of $\mathcal{G}!$ together with lemma 5.3 imply the existence of mapping $!\colon [\![\,\mathsf{PB}([\![\,\mathcal{R}\,]\!])\,]\!] \to [\![\,\mathcal{G}!\,]\!]$. On the other hand, we have a mapping $[g!_1, g!_2]\colon [\![\,\mathcal{G}!\,]\!] \to [\![\,S_1\,]\!] \times [\![\,S_2\,]\!]$. Since the diamond in Fig. 2(a) is commutative, and the set $\mathsf{PB}([\![\,\mathcal{R}\,]\!])$ contains *all* pairs in $[\![\,S_1\,]\!] \times [\![\,S_2\,]\!]$ for which we have commutativity, the image of mapping $[g!_1, g!_2]$ will be inside $\mathsf{PB}([\![\,\mathcal{R}\,]\!])$. Thus, in fact we have a mapping $[g!_1, g!_2]\colon [\![\,\mathcal{G}!\,]\!] \to \mathsf{PB}([\![\,\mathcal{R}\,]\!])$. By commutativity of the diagrams involved, these mappings are mutually inverse and hence $\mathsf{PB}([\![\,\mathcal{R}\,]\!]) \cong [\![\,\mathcal{G}!\,]\!]$. Lemma 5.3 now implies $[\![\,\mathcal{S}_\Sigma\,]\!] \cong [\![\,\mathcal{G}!\,]\!]$ Q.E.D.

The result of Theorem 5.1 can be presented in the following (typically categorical) way.

5.4 Corollary. Let ***SchemaSpan***, ***SetSpan***, ***SchemaCospan*** and ***SetCospan*** denote universes of spans and cospans in the universes of schemas and sets, respectively. Then the following diagram is commutative:

$$\begin{array}{rccc}
\text{Syntax:} & \boldsymbol{SchemaSpan} & \xrightarrow{\mathsf{merge} \Rrightarrow \mathsf{norm}} & \boldsymbol{SchemaCospan} \\
\text{instanceSet}\downarrow & [\![\,.\,]\!]\downarrow & & \downarrow[\![\,.\,]\!] \\
\text{Semantics:} & \boldsymbol{SetCospan} & \xrightarrow[\mathsf{Merge}]{} & \boldsymbol{SetSpan}
\end{array}$$

Here Merge denotes the operation of semantic merge (Definition 3.2) up to IC-equivalence (Proposition 3.3), merge is the operation of syntactic merge and norm is the operation of schema normalization (Section 4.2) also determined up to IC-equivalence. Commutativity of this diagram is a formal explication of the duality between syntax and semantics, and a formal semantic justification of syntactical integration algorithms.

References

[1] S. Alagic and P. Bernstein. A model theory for generic schema management. In *Proc. DBPL'2001*, 2001.

[2] M. Barr and C. Wells. *Toposes, Triples, Theories.* Springer, 1985.

[3] P. Bernstein. Applying model management to classical metadata problems. In *Proc. CIDR'2003*, pages 209–220, 2003.

[4] P. Bernstein, A. Halevy, and R. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[5] B. Cadish and Z. Diskin. Heterogenious view integration via sketches and equations. In *Foundations of Intelligent Systems, 9th Int.Symposium*, Springer LNAI #1079, pages 603–612, 1996.

[6] Z. Diskin and B. Kadish. A graphical yet formalized framework for specifying view systems. In *Advances in Databases and Information Systems*, pages 123–132, 1997. ACM SIGMOD Digital Anthology: vol.2(5), ADBIS'97.

[7] Z. Diskin and B. Kadish. Generic model management. In Rivero, Doorn, and Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 258–265. Idea Group, 2005.

[8] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. Int. Conf. on Database Theory, ICDT*, 2003.

[9] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1), 2005.

[10] M. Lenzerini. Data integration: A theoretical perspective. (Invited tutorial). In *21st ACM Symposium on Principles of database systems*, pages 233–246, 2002.

[11] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *27th VLDB Conference, Roma,Italy*, 2001.

[12] E. Manes. *Algebraic Theories.* No.26 in Graduate Text in Mathmetics. Springer Verlag, 1976.

[13] S. Melnik, P. Bernstein, A. Halevy, and E. Rahm. Supporting executable mappings in model management. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 167–178, New York, NY, USA, 2005. ACM Press.

[14] R. Miller. Using schematically heterogeneous structures. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 189–200, New York, NY, USA, 1998. ACM Press.

[15] R. Miller, Y. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In R. Agrawal, S. Baker, and D. Bell, editors, *19th International Conference on Very Large Data Bases*, pages 120–133, 1993.

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

[17] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[18] Y. Velegrakis, R. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *Proc. 29th VLDB*, 2003.