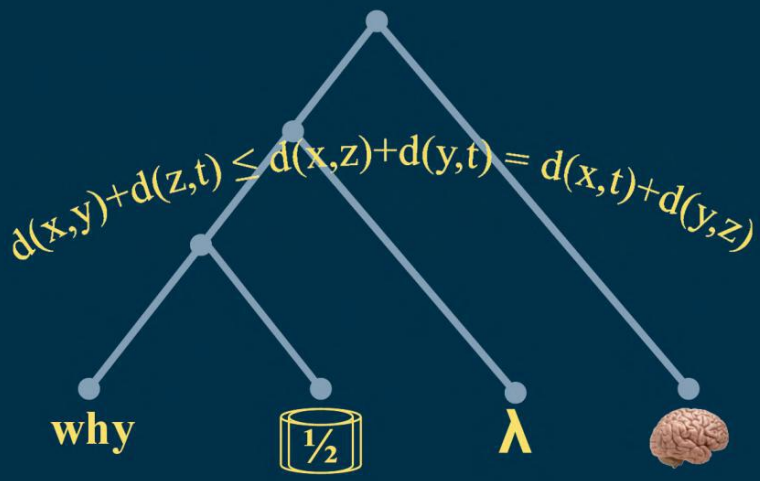
Festschrift

LNCS 8000

Val Tannen Limsoon Wong
Leonid Libkin Wenfei Fan
Wang-Chiew Tan Michael Fourman (Eds.)

In Search of Elegance in the Theory and Practice of Computation

Essays Dedicated to Peter Buneman



Springer

Val Tannen Limsoon Wong
Leonid Libkin Wenfei Fan
Wang-Chiew Tan Michael Fourman (Eds.)

In Search of Elegance in the Theory and Practice of Computation

Essays Dedicated to Peter Buneman

Volume Editors

Val Tannen
University of Pennsylvania, Department of Computer and Information Science
3330 Walnut Street, Philadelphia, PA 19104, USA
E-mail: val@cis.upenn.edu

Limsoon Wong
National University of Singapore, School of Computing
13 Computing Drive, Singapore 117417, Singapore
E-mail: wongls@comp.nus.edu.sg

Leonid Libkin
Wenfei Fan
Michael Fourman
The University of Edinburgh, School of Informatics
10 Crichton Street, Edinburgh EH8 9AB, UK
E-mail: {libkin; wenfei@inf.ed.ac.uk}, michael.fourman@ed.ac.uk

Wang-Chiew Tan
University of California, Department of Computer Science
1156 High Street, Santa Cruz, CA 95064, USA
E-mail: wctan@cs.ucsc.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-41659-0 e-ISBN 978-3-642-41660-6
DOI 10.1007/978-3-642-41660-6
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: : 2013951160

CR Subject Classification (1998): H.2, D.3, H.3, F.3, F.4.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Table of Contents

Models for Data-Centric Workflows*

Serge Abiteboul[1] and Victor Vianu[2]

[1] INRIA Saclay
[2] UC San Diego and INRIA Saclay

Abstract. We present two models for data-centric workflows: the first based on business artifacts and the second on Active XML. We then compare the two models and argue that Active XML is strictly more expressive, based on a natural semantics and choice of observables. Finally, we mention several verification results for the two models.

1 Introduction

Workflows and database systems are two essential software components that often have difficulties interoperating. Data-centric workflow systems alleviate this problem by providing an integrated approach to data management and workflows. They allow the management of data evolution by tasks with complex sequencing constraints, as encountered for instance in scientific workflow systems, information manufacturing systems, e-government, e-business or healthcare global systems.

Data-centric workflows have evolved from process-centric formalisms, which traditionally focus on control flow while under-specifying the underlying data and its manipulations by the process tasks, often abstracting them away completely. In contrast, data-aware formalisms treat data as first-class citizens. A notable exponent of this class is the *business artifact model* pioneered in [17], deployed by IBM in commercial products and consulting services, and further studied in a line of follow-up works [4 6 9 10 5 15 13 14]. Business artifacts (or simply "artifacts") model key business-relevant entities that evolve in response to events in their life-cycle. See [11] for a brief survey on the topic.

Another effort at modeling data-centric workflows relies on Active XML (AXML). An AXML document consists of an XML document with embedded function calls, modeling tasks in the workflow. Each call generates a data-carrying task which in turn can spawn additional sub-tasks. The functions are specified using queries based on tree patterns [3 1]. See [2] for a discussion on how Active XML can serve as a workflow model.

Business artifacts and AXML provide two different paradigms for specifying data-centric workflows. A natural question concerns their relative expressive power. We describe a semantics introduced in [7] for comparing the expressiveness of workflow systems relative to a set of observables, and argue that Active XML is strictly more expressive than the variant of business artifacts presented here.

* This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513. http://webdam.inria.fr/

V. Tannen et al. (Eds.): Buneman Festschrift, LNCS 8000, pp. 1–12, 2013.

Several recent works have considered the problem of verifying business artifacts [8, 7] and Active XML systems [3]. The verification problem consists of statically checking whether all runs satisfy desirable properties expressed in an extension of linear-time temporal logic (LTL). The presence of data results in a challenging infinite-state verification problem, due to the infinite data domain. Rather than relying on general-purpose software verification tools suffering from well-known limitations, the above works address this problem by identifying relevant classes of business artifacts and Active XML systems for which fully automatic verification is possible. We briefly summarize these results.

2 The Business Artifact Model

We describe a minimalistic variant of the business artifact model, adequate for conveying the flavor of the approach. The presentation is informal, relying mainly on a running example (the formal development is provided in [8, 7]). The example models an e-commerce business process in which the customer chooses a product and a shipment method and applies various kinds of coupons to the order. After the order is filled, the system awaits for the customer to submit a payment. If the payment matches the amount owed, the system proceeds to shipping the product.

In the minimalistic model, an artifact is simply an evolving record of values. The values are referred to by variables (sometimes called *attributes*). In general, an artifact system consists of several artifacts, evolving under the action of *services*, specified by pre- and post-conditions. For simplicity, we use a single artifact with the following variables

`status`, `prod_id`, `ship_type`, `coupon`, `amount_owed`,
`amount_paid`, `amount_refunded`.

The `status` variable tracks the status of the order and can take values such as "edit_product", "received_payment", "shipping", "canceling", etc. Thus, `status` can be viewed as recording the current stage of the order processing. In conjunction with pre-and-post conditions of services, this allows simulating a classical form of sequencing based on finte-state automata. However, unlike classical process-centric approaches, the sequencing can also depend on properties of the *data*.

The artifact system is equipped with a database including the following tables, where underlined attributes denote keys. Recall that a key is an attribute that uniquely identifies each tuple in a relation.

`PRODUCTS`(<u>`id`</u>, `price`, `availability`, `weight`),
`COUPONS`(<u>`code`</u>, `type`, `value`, `min_value`, `free_shiptype`),
`SHIPPING`(<u>`type`</u>, `cost`, `max_weight`),
`OFFERS`(<u>`prod_id`</u>, `discounted_price`, `active`).

The database also satisfies the following foreign keys:

`COUPONS`[`free_shiptype`] $\subseteq$ `SHIPPING`[`type`] and
`OFFERS`[`prod_id`] $\subseteq$ `PRODUCTS`[`id`].

The starting configuration of every artifact system is constrained by an initialization condition, which here states that `status` initialized to "edit_prod", and all other variables to "undefined". By convention, we model undefined variables using the reserved constant λ.

The Services. Recall that artifacts evolve under the action of services. Each service is specified by a pre-condition π and a postcondition ψ, both existential first-order ($\exists$FO) sentences. The pre-condition refers to the current values of the artifact variables and the database. The post-condition ψ refers simultaneously to the current and *next* artifact values, as well as the database. In addition, both π and ψ may use arithmetic constraints on the variables, limited to linear inequalities over the rationals.

The following services model two of the business process tasks of the example. We use primed artifact variables x' to refer to the *next* value of variable x.

choose_product. The customer chooses a product.

$$\pi : \texttt{status} = \textit{"edit\_prod"}$$
$$\psi : \exists p, a, w(\texttt{PRODUCTS}(\texttt{prod\_id}', p, a, w) \wedge a > 0)$$
$$\wedge \texttt{status}' = \textit{"edit\_shiptype"}$$

choose_shiptype. The customer chooses a shipping option.

$$\pi : \texttt{status} = \textit{"edit\_ship"}$$
$$\psi : \exists c, l, p, a, w(\texttt{SHIPPING}(\texttt{ship\_type}', c, l) \wedge$$
$$\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge l > w) \wedge$$
$$\texttt{status}' = \textit{"edit\_coupon"} \wedge \texttt{prod\_id}' = \texttt{prod\_id}$$

Notice that the pre-conditions of the services check the value of the `status` variable. For instance, according to **choose_product**, the customer can only input her product choice while the order is in "edit_prod" status.

Also notice that the post-conditions constrain the next values of the artifact variables (denoted by a prime). For instance, according to **choose_product**, once a product has been picked, the next value of the status variable is "edit_shiptype", which will at a subsequent step enable the **choose_shiptype** service (by satisfying its pre-condition). The interplay of pre- and post-conditions achieves a sequential filling of the order, starting from the choice of product and ending with the claim of a coupon. A post-condition may refer to both the current and next values of the artifact variables. For instance, consider the service **choose_shiptype**. The fact that only the shipment type is picked while the product remains unchanged, is modeled by preserving the product id: the next and current values of the corresponding artifact variable are set equal.

Pre- and post-conditions may query the database. For instance, consider the function **choose_product**. The post-condition ensures that the product id chosen by the customer is that of an available product (by checking that it appears in a `PRODUCTS` tuple, whose availability attribute is positive).

Semantics. The semantics of an artifact system consists of its *runs*. Given a database D, a run is an infinite sequence $\{\rho_i\}_{\geq 0}$ of artifact records such that ρ_0 and D satisfy

the initial condition of the system, and for each $i \geq 0$ there is a service S of the system such that ρ_i and D satisfy the pre-condition of S and ρ_i, ρ_{i+1} and D satisfy its post-condition. For uniformity, blocking prefixes of runs are extended to infinite runs by repeating forever their last record.

We note that the full business artifact model is still in flux. In its current state (e.g., see [12]), the model allows artifact attributes containing collections, rather than just atomic atoms. It also provides richer forms of control, achieved by a *hierarchy* of services.

3 Active XML Workflows

We next describe the specification of workflows in Active XML. We use a model called *Guard Active XML* (GAXML for short) [3, 7].

GAXML documents are abstractions of XML with embedded service calls. A GAXML document is a forest of unordered, unranked trees, whose internal nodes are labeled with tags from a finite alphabet and whose leaves are labeled with tags, data values, or function symbols. More precisely, a function symbol $!f$ indicates a node where function f can be called, and a function symbol $?f$ indicates that a call to f has been made but the answer has not yet been returned. For example, a GAXML document is shown in Figure 1.

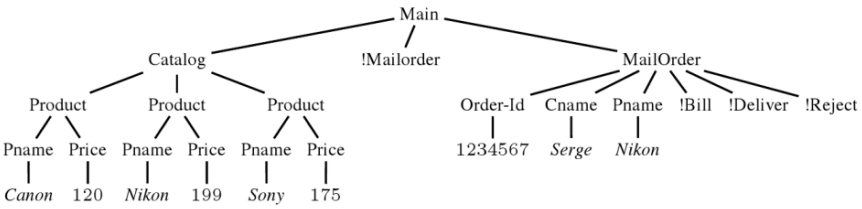


Fig. 1. A GAXML document

The GAXML document may be subject to constraints specified by a DTD, as well as Boolean combinations of tree patterns. For example, the negation of the pattern in Figure 3(a) says that an Order ID uniquely determines the product and customer names. In patterns, double edges denote descendant and single edges the child relation.

A GAXML document evolves as a result of making function calls and receiving their results. A call can be made at any point, as long as a specified pre-condition, called a *call guard*, is satisfied. The argument of the call is specified by a query on the document, producing a forest. Both the call guard and input query may refer to the node at which the call is made (denoted *self*), so the location of the call in the document is important. The result of a function call consists of another GAXML document, so a forest, whose trees are added as siblings of the node x where the call was made. After the answer of a call at node x is returned, the call may be kept or the node x may be deleted. This is specified by the schema, for each function. If calls to $!f$ are kept, f is called *continuous*, otherwise it is *non-continuous*.

For example, consider the `MailOrder` function in Figure 1. Intuitively, its role is to fetch new mail orders from customers. For instance, one result of a call to the function `!MailOrder` may consist of the subtree with root `MailOrder` in Figure 1. Since new orders should be fetched indefinitely, the call `!MailOrder` is maintained after each result is returned, so `MailOrder` is specified to be continuous. On the other hand, consider the function `!Bill` occurring in a `MailOrder`. This is meant to be called only once, in order to carry out the billing task. Once the task is finished, the call can be removed. Therefore, `Bill` is specified as a non-continuous function.

Consider again the function `MailOrder`, whose role is to fetch new orders from external users or services. Since the function is processed externally, the semantics of its evaluation is not known. We call such a function *external*. Its specification consists only of its call guard and input query, and its answer is only constrained by signature information provided by the schema. In addition to external functions, there are functions processed internally by the GAXML system. These are called *internal*. For example, `Bill` is such a function. When a call to `Bill` is made at a node x labeled `!Bill`, the label of x turns to `?Bill` (to indicate that a call has been made whose answer is still pending) and the call is processed internally. Specifically, the call generates a new GAXML document (a *running call*) that evolves until it satisfies a condition called *return guard*. Intuitively, the return guard indicates that the task corresponding to the call has been completed and the result can be returned. The contents of the result is specified by a *return query*. For example, the answer to a call to `Bill` can be returned once payment has been received. The answer, specified by the return query, provides the product paid for and amount of payment (see Example 1).

Once the result of a call has been returned, the GAXML document of the completed running call is removed. In order for the result to be returned at the correct location (next to node x), a mapping called *eval* is maintained between nodes where calls have been made and GAXML document corresponding to the running call (e.g., see Figure 2). The system evolves by repeated function calls and answer returns, occurring one at a time non-deterministically. This may reach a *blocking instance* in which no function can be called and no result can be returned, or may continue forever, leading to an infinite run. For example, runs of the Mail Order system are always infinite since new mail orders can always be fetched. For uniformity, we make all runs infinite by repeating blocking instances forever.

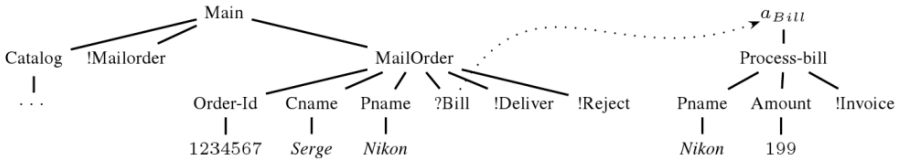


Fig. 2. An instance with an *eval* link

Note that call guards provide a very useful form of control. In particular, they are instrumental in enforcing desired ordering among tasks. For instance, in the Mail Order example, to enforce that delivery of a product can only occur after billing has been completed, it is sufficient for the call guard of `!Deliver` to check that neither `!Bill` nor `?Bill` occur in the subtree corresponding to the order.

Example 1. The function `Bill` used in Figure 1 is specified as follows. It is internal and non-continuous. Its call guard is the pattern in Figure 3 (b), checking that the ordered product is available. The input query is the query in Figure 4. Assuming that `Invoice` is an external function eventually returning `Payment` (with product and amount paid), the return guard and return query of `Bill` are shown in Figure 5.

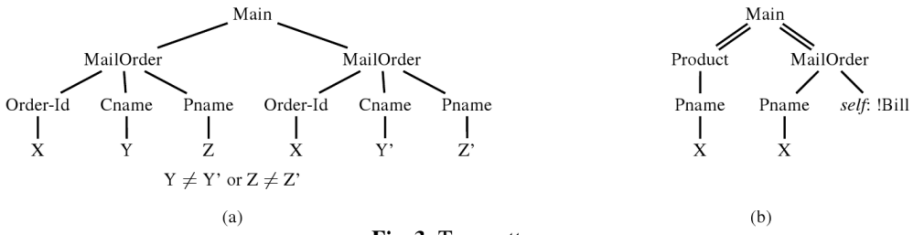


Fig. 3. Two patterns

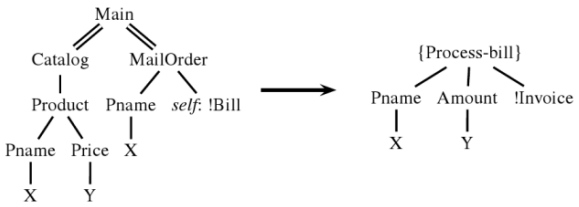


Fig. 4. Argument query for `!Bill`

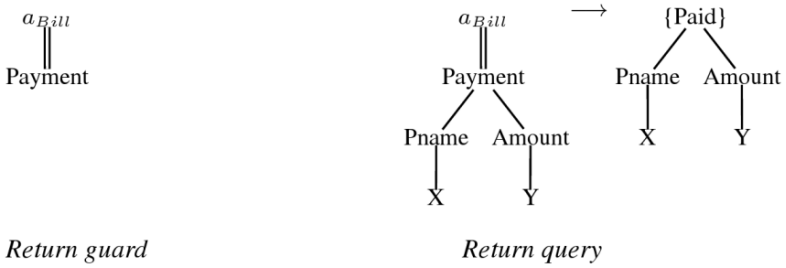


Fig. 5. Return guard and query for `!Bill`

In GAXML, workflow control is provided by the guards associated with functions. There are many other possible ways to control sequencing of tasks. In [7], the following alternative workflow control mechanisms are also discussed:

Automata. The automata are non-deterministic finite-state transition systems, in which states have associated tree pattern formulas with free variables acting as parameters. A transition into a state can only occur if its associated formula is true. In addition, the automaton may constrain the values of the parameters in consecutive states.

Temporal Properties. These are expressed in a temporal logic with tree patterns and Past LTL operators. A temporal formula constrains the next instance based on the history of the run.

Subject to some minor technical assumptions, it is shown in [7] that the power of guards, automata, and temporal logic as workflow specification mechanisms is the same. More surprisingly, static constraints alone can largely simulate all three control mechanisms.

4 Comparing Business Artifacts and Active XML Workflows

We have discussed two models of data-centric workflows: business artifacts and Active XML. A natural question is whether their expressiveness can be measured and compared. The models are quite different in their representation of data and events, so a direct comparison is meaningless. In [1], a framework is developed for comparing workflow specification languages, by mapping different models to a common abstraction using the notion of *workflow view*. Depending on the specific needs, a workflow view might retain information about some abstract state of the system and its evolution, about some particular events and their sequencing, about the entire history of the system so far, or a combination of these and other aspects. Even if not made explicit, a view is often the starting point in the design of workflow specifications. This further motivates using views to bridge the gap between different specification languages.

To see how this might be done, consider a workflow W specified by tasks and pre/post conditions and another workflow W' specified as a state-transition system, both pertaining to the same application. One way to render the two workflows comparable is to define a view of W as a state-transition system compatible with W'. This can be done by defining states using queries on the current instance and state transitions induced by the tasks. To make the comparison meaningful, the view of W should retain in states the information relevant to the semantics of the application, restructured to make it compatible with the representation used in W'. More generally, views may be used to map given workflows models to an entirely different model appropriate for the comparison. In [1], the general notion of workflow view is defined and a form of bisimulation over views is introduced to capture the fact that one workflow simulates another. The bisimulation applies to the *tree of runs* of the systems to be compared.

Using the framework based on views, it is shown in [1] that Active XML is strictly more expressive than business artifacts (without arithmetic and data depedencies). Specifically, Active XML can simulate business artifacts, but the converse is false.

The first result uses views mapping XML to relations and functions to services, so that artifacts become views of Active XML systems. For the negative result we use views retaining just the trace of function and service calls from the Active XML and the artifact system. This is a powerful result, since it extends to *any* views exposing *more* information than the function/service traces.

5 Verification

The verification problem for business artifacts as well as Active XML workflows has been considered in several recent works [3, 8, 7]. The problem consists of checking, for a given workflow specification and temporal property, whether all runs of the workflow system satisfy the property. For instance, one may want to verify whether some static property (e.g., all ordered products are available) and some dynamic property (e.g. an order is never delivered before payment is received) always hold. The temporal properties are specified in extensions of LTL, linear-time temporal logic. The presence of an unbounded data domain yields a challenging infinite-state verification problem.

In order to specify temporal properties we use an extension of LTL. Recall that LTL is propositional logic augmented with temporal operators such as **G** (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [18]). For example, $\mathbf{G}p$ says that p holds at all times in the run, $\mathbf{F}p$ says that p will eventually hold, and $\mathbf{G}(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future. In order to take into account data, we consider extensions of LTL in which propositions are interpreted by statements on current snapshots of the system. The language used to express the statements is dependent on the particular data model. For business artifacts, the language is FO, yielding the extension LTL(FO). For Active XML, the language consists of tree patterns, yielding LTL(*Tree*). We consider each model in turn.

Verification for Business Artifacts. For business artifacts, propositions are interpreted as quantifier-free FO formulas using current and next artifact values, constants, and the database. For example, suppose we wish to specify the property that if a correct payment is submitted then at some time in the future either the product is shipped or the customer is refunded the correct amount. The property is of the form $\mathbf{G}(p \rightarrow \mathbf{F}q)$, where p says that a correct payment is submitted and q states that either the product is shipped or the customer is refunded the correct amount. Moreover, if the customer is refunded, the amount of the correct payment (given in p) should be the same as the amount of the refund (given in q). This requires using a global variable x in both p and q. More precisely, p is interpreted as the formula $\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}$ and q as $\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x$. This yields the LTL(FO) property

$$\forall x \mathbf{G}((\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}) \rightarrow \mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x))$$

Note that, as one would expect, the global variable x is universally quantified at the end.

For artifact systems and properties without arithmetic constraints or data dependencies it was shown that verification is decidable [8]. The complexity is PSPACE-complete for a fixed number of attributes, and EXPSPACE otherwise. This is the best one can expect, given that even very simple static analysis problems for finite-state systems are already PSPACE-complete.

It turns out that the verification algorithm can be extended to specifications and properties that use a *total order* on the data domain, which is useful in many cases. This however complicates the algorithm considerably, since the order imposes global constraints on runs. The verification algorithm was first extended in [8] for the case of a dense countable order with no end-points (such as the rationals). This was later generalized to an arbitrary total order by Segoufin and Torunczyk [16] using automata-theoretic techniques. In both cases, the worst-case complexity remains PSPACE.

Unfortunately, the above decidability result fails even in the presence of simple data dependencies or arithmetic. As shown in [8 7], verification becomes undecidable as soon as the database has at least one key dependency, *or* if the specification of the artifact system uses simple arithmetic constraints allowing to increment and decrement by one the value of some atributes. Therefore, a restriction is imposed in [7] to achieve decidability.

The restriction is designed to limit the data flow between occurrences of the same artifact attribute throughout runs of the system that satisfy the desired property. As a first cut, a possible restriction would prevent any data flow path between unequal occurrences of the same artifact attribute. Let us call this restriction *acyclicity*. While acyclicity would achieve the goal of rendering verification decidable, it is too strong for many practical situations. In the example of Section 2, a customer can choose a shipping type and coupon and repeatedly change her mind and start over. Such repeated performance of a task is useful in many scenarios, but would be prohibited by acyclicity of the data flow.

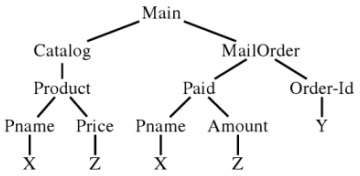
To this end, we define in [7] a more permissive restriction called *feedback freedom*. Intuitively, paths among different occurrences of the same attribute are permitted, but only as long as each value of the attribute is independent on its previous values. This is ensured by a syntactic condition that takes into account both the artifact system and the property to be verified. We omit here the rather technical details. It is shown in [7] that feedback freedom of an artifact system together with an LTL(FO) property can be checked in PSPACE by reduction to a test of emptiness of a two-way alternating finite-state automaton. Feedback freedom turns out to ensure decidability of verification in the presence of linear constraints, and also under a large class of data dependencies including keys and foreign keys.

Verification of Active XML Workflows. Properties of Active XML workflows are expressed in LTL(*Tree*), an extension of LTL in which propositions are interpreted by tree patterns. For example, suppose that we wish to verify the following property:

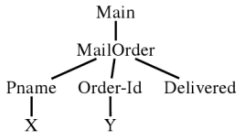
Every product for which a correct amount has been paid is eventually delivered.

To formulate the property, we use tree patterns with variables binding to data values (without going into details, let us denote such a language of tree patterns by *Tree*).

The above property can be expressed in the language LTL(*Tree*) as follows. We start out with the LTL formula $\mathbf{G}(p \rightarrow \mathbf{F}q)$. The proposition p is replaced by the tree pattern



checking that the payment received for product X of order Y is in the right amount Z. The proposition q is replaced by the tree pattern



checking that product X of the same order Y is eventually delivered. Note that we wish X and Y to be the same in the tree patterns for p and q, so these are globally quantified; in contrast, Z is locally quantified. The resulting LTL(*Tree*) formula is shown in Figure 6.

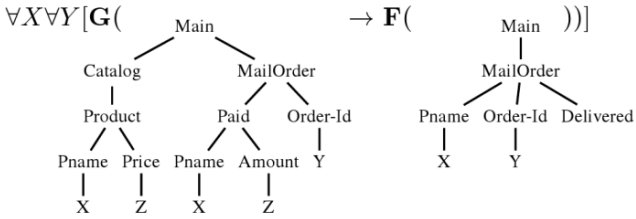


Fig. 6. An LTL(*Tree*) formula

It is shown in [3] that verification of LTL(*Tree*) properties of Active XML workflows is decidable in 2-EXPTIME, under a syntactic restriction ensuring that the workflow has only runs of bounded length.

6 Conclusion

Data-centric workflows are increasingly prevalent and there is a need for high-level models and languages for specifying and reasoning about them. In this note, we presented two such models: business artifacts (initiated at IBM Research), and Active XML (developed at INRIA). In both models, data is a first-class citizen, and it evolves as a result of events in its life cycle. However, there are significant differences in the two approaches. The data in business artifacts is relational, while in Active XML it is an extension of XML. Events in the life-cycle are modeled in business artifacts by services specified by pre-and-post conditions, while Active XML models events by function

calls embedded in the data. To compare such distinct models, we proposed an approach based on workflow views that map different models to a common abstraction, and a notion of bisimulation on the trees of runs of the abstracted systems. Using this framework, we showed that Active XML is strictly more expressive than business artifacts (for the variants presented here). This is not suprising given that Active XML is a much richer model. A more detailed discussion of the ability of Active XML to capture the facets of an artifact model, as informally described in [17], is presented in [2], where it is argued that Active XML can in fact capture all aspects of the artifact approach. Moreover, the notions of subtask and of collection of artifacts are naturally built into the model, whereas the business artifact model as in [8,7] has to be extended in order to model them. Such extensions are indeed discussed in [12].

We finally reviewed some recent results on the automatic verification of workflows in both languages. These suggest that automatic verification may be feasible for a practically significant class of workflows and properties.

References

1. Abiteboul, S., Bourhis, P., Vianu, V.: Comparing workflow specification languages: A matter of views. In: ICDT (2011)
2. Abiteboul, S., Bourhis, P., Galland, A., Marinoiu, B.: The axml artifact model. In: TIME, Symposium on Temporal Representation and Reasoning, pp. 11–17 (2009)
3. Abiteboul, S., Segoufin, L., Vianu, V.: Static analysis of active XML systems. ACM Trans. Database Syst. 34(4) (2009)
4. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifact-centered operational modeling: Lessons from customer engagements. IBM Systems Journal 46(4), 703–721 (2007)
5. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Proc. Int. Conf. on Business Process Management (BPM), pp. 288–304 (2007)
6. Bhattacharya, K., et al.: A model-driven approach to industrializing discovery processes in pharmaceutical research. IBM Systems Journal 44(1), 145–162 (2005)
7. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. In: ICDT (2011)
8. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: ICDT, pp. 252–267 (2009)
9. Gerede, C.E., Bhattacharya, K., Su, J.: Static analysis of business artifact-centric operational models. In: IEEE International Conference on Service-Oriented Computing and Applications (2007)
10. Gerede, C.E., Su, J.: Specification and verification of artifact behaviors in business process models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007), http://www.springerlink.com/content/c371144007878627
11. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: OTM Conferences (2), pp. 1152–1163 (2008)
12. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of 7th Intl. Workshop on Web Services and Formal Methods, WS-FM (2010)

13. Kumaran, S., Liu, R., Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
14. Küster, J.M., Ryndina, K., Gall, H.C.: Generation of business process models for object life cycle compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
15. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
16. Segoufin, L., Torunczyk, S.: Automata based verification over linearly ordered data domains. In: Int'l. Symp. on Theoretical Aspects of Computer Science, STACS (2011)
17. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
18. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)

Relational Databases and Bell's Theorem

Samson Abramsky

University of Oxford
samson.abramsky@cs.ox.ac.uk

Abstract. Our aim in this paper is to point out a surprising formal connection, between two topics which seem on face value to have nothing to do with each other: relational database theory, and the study of non-locality and contextuality in the foundations of quantum mechanics. We shall show that there is a remarkably direct correspondence between central results such as Bell's theorem in the foundations of quantum mechanics, and questions which arise naturally and have been well-studied in relational database theory.

1 Introduction

Our aim in this paper is to point out a surprising formal connection, between two topics which seem on face value to have nothing to do with each other:

- Relational database theory.
- The study of non-locality and contextuality in the foundations of quantum mechanics.

We shall show, using the unified treatment of the latter developed in [3], that there is a remarkably direct correspondence between central results such as Bell's theorem in the foundations of quantum mechanics, and questions which arise naturally and have been well-studied in relational database theory.

In particular, we shall see that the question of whether an "empirical model", of the kind which can be obtained by making observations of measurements performed on a physical system, admits a classical physical explanation in terms of a local hidden variable model, is mathematically equivalent to the question of whether a database instance admits a universal relation. The content of Bell's theorem and related results is that there are empirical models, predicted by quantum mechanics and confirmed by experiment, which do not admit such a universal relation. Moreover, while the original formulation of Bell's theorem involved probabilities, there are "probability-free" versions, notably Hardy's construction, which correspond directly to relational databases.

In fact, we shall show more broadly that there is a common mathematical language which can be used to described the key notions of both database theory, in the standard relational case and in a more general "algebraic" form covering e.g. a notion of probabilistic databases, and also of the theory of non-locality and contextuality, two of the key quantum phenomena. These features are central to

V. Tannen et al. (Eds.): Buneman Festschrift, LNCS 8000, pp. 13–35, 2013.

current discussions of quantum foundations, and provide non-classical resources for quantum information processing.

The present paper is meant to be an introduction to these two topics, emphasizing their common content, presented in a manner which hopefully will be accessible to readers without prior knowledge of either.

How should this unexpected connection be interpreted? One idea is that the notion of contextuality is rather fundamental, and we can see some outlines of a common 'logic of contextuality' arising from this appearance of common structure in very different settings.

Ideally, some deeper connections can also be found, leading to interesting transfers of results and methods. A first step in this direction has already been taken, in joint work with Georg Gottlob and Phokion Kolaitis [4], in the closely related field of constraint satisfaction. An algorithmic question which arises naturally from the quantum side (see [2]) leads to a refined version of the constraint satisfaction paradigm, *robust constraint satisfaction*, and to interesting new complexity results.

2 Relational Databases

2.1 Review of Basic Notions

We shall begin by reviewing some basic notions of relational database theory.

We start with an example to show the concrete scenario which is to be formalized.

Example. Consider the following data table:

branch-name	**account-no**	**customer-name**	**balance**
Cambridge	10991-06284	Newton	£2,567.53
Hanover	10992-35671	Leibniz	€11,245.75
...	...	...	...

Let us anatomize this table. There are a set of *attributes*,

$$\{\textbf{branch-name}, \textbf{account-no}, \textbf{customer-name}, \textbf{balance}\}$$

which name the columns of the table. The entries in the table are 'tuples' which specify a value for each of the attributes. The table is a set of such tuples. A database will in general have a set of such tables, each with a given set of attributes. The *schema* of the database — a static, syntactic specification of the kind of information which can reside in the database — is given by specifying the set of attributes for each of the tables. The state of the database at a given time will be given by a set of tuples of the appropriate type for each of the tables in the schema.

We now proceed to formalize these notions.

We fix some set $\mathcal{A}$ which will serve as a universe of attributes. A database schema Σ over $\mathcal{A}$ is a finite family $\Sigma = \{A_1, \ldots, A_k\}$ of finite subsets of $\mathcal{A}$.

At this — surprisingly early! — point, we come to an interesting juncture. There are two standard approaches to formalising the notion of relation which can be found in the relational database literature. One — the 'unnamed perspective' [1] — is to formalize the notion of tuple as an ordered n-tuple in D^n for some set D of data values; a relation is then a subset of D^n. This is motivated by the desire to make the connection to the standard notion of relational structure in first-order logic as direct as possible. This choice creates a certain distance between the formal notion of relation, and the informal notion of table; in practice this is not a problem.

For our purposes, however, we wish to make a different choice — the 'named perspective' [1]: we shall formalize the notion of tuple, and hence of relation, in a fashion which directly reflects the informal notion. As we shall see, this will have both mathematical and conceptual advantages for our purposes. At the same time, there is no real problem in relating this formalism to the alternative one found in the literature. Note that the style of formalization we shall use is also commonly found in the older literature on relational databases, see e.g. [26].

We shall assume that for each $a \in \mathcal{A}$ there is a set D_a of possible data values for that attribute. Thus for example the possible values for **customer-name** should be character strings, perhaps with some lexical constraints; while for **balance** the values should be pairs (**currency**, **amount**), where **currency** comes from some fixed list (£, €, ...), and **amount** is a number. These correspond to *domain integrity constraints* in the usual database terminology.

Given $A \in \Sigma$, we define the set of A-tuples to be $\prod_{a \in A} D_a$. Thus an A-tuple is a function which assigns a data value in D_a to each $a \in A$.

In our example above, the first tuple in the table corresponds to the function

$$\{\textbf{branch-name} \mapsto \text{Cambridge}, \textbf{account-no} \mapsto 10991{-}06284,$$

$$\textbf{customer-name} \mapsto \text{Newton}, \textbf{balance} \mapsto £2,567.53\}$$

A relation of type A is a finite set of A-tuples. Given a schema Σ, an *instance* of the schema, representing a possible state of the database, is given by specifying a relation of type A for each $A \in \Sigma$.

Operations on Relations. We consider some of the fundamental operations on relations, which play a central rôle in relational databases. Firstly, relations of type A live in the powerset $\mathcal{P}(\prod_{a \in A} D_a)$, which is a boolean algebra; so boolean operations such as union, intersection, and set difference can be applied to them.

Note that the set of data values may in general be infinite, whereas the relations considered in database theory are finite. Thus one must use set difference rather than an 'absolute' notion of set complement.

Next, we consider the operation of projection. In the language of A-tuples, projection is *function restriction*. That is, given an A-relation R, and a subset $B \subseteq A$, we define:

$$R|_B := \{t|_B \ : \ t \in R\}.$$

Here, since $t \in \prod_{a \in A} D_a$, $t|_B$ just means restriction of the function t to B, which is a subset of its domain. This operation is then lifted pointwise to relations.

Now we consider the independent combination of relations, which is cartesian product in the standard formalism. The representation of tuples as functions leads to a 'logarithmic shift' in the representation[1], whereby this operation is represented by *disjoint union* of attribute sets.

Given an A-relation R and a B-relation S, we form the disjoint union $A \sqcup B$, and the $A \sqcup B$-relation

$$R \otimes S \ := \ \{t \in \prod_{a \in A \sqcup B} D_a \ : \ t|_A \in R \ \wedge \ t|_B \in S\}.$$

Of course, as concrete sets A and B may overlap. We can force them to be disjoint by 'tagging' them appropriately, e.g.

$$A \sqcup B \ := \ \{0\} \times A \ \cup \ \{1\} \times B.$$

The minor housekeeping details of such tagging can safely be ignored.[2] We shall henceforth do so without further comment.

This is only a subset of the operations available in standard relational algebra [26]. A more complete discussion could be given in the present setting, but this will suffice for our purposes.

2.2 The Functorial View

We shall now show how the relational database formalism, in the style we have developed it, has a direct expression in functorial terms. This immediately brings a great deal of mathematical structure into play, and will allow us to relate some important database notions to concepts of much more general standing.

We shall assume the rudiments of the language of categories, functors and natural transformations. All the background we shall need is covered in the charming (and succinct) text [25].

We shall consider the partial order **Att** of finite subsets of $\mathcal{A}$, ordered by inclusion, as a category.

We shall define a functor $\mathcal{T} \ : \ \mathbf{Att}^{\mathbf{op}} \longrightarrow \mathbf{Set}$ where $\mathcal{T}(A)$ is the set of A-tuples. Formally, we define

$$\mathcal{T}(A) \ := \ \prod_{a \in A} D_a,$$

and if $A \subseteq B$, we define the *restriction map* $\rho^B_A : \mathcal{T}(B) \longrightarrow \mathcal{T}(A)$ by

$$\rho^B_A : t \mapsto t|_A.$$

[1] Think of $2^a 2^b = 2^{a+b}$, and hence $\log(xy) = \log(x) + \log(y)$.

[2] The relevant result is the coherence theorem for monoidal categories [20].

It is easy to verify functoriality of $\mathcal{T}$, which means that, whenever $A \subseteq B \subseteq C$,

$$\rho^B_A \circ \rho^C_B = \rho^C_A,$$

and also that $\rho^A_A = \mathsf{id}_A$. Thus $\mathcal{T}$ is a *presheaf*, and restriction is exactly function restriction.

We also have the covariant powerset functor $\mathcal{P} : \mathbf{Set} \longrightarrow \mathbf{Set}$, which acts on functions by direct image: if $f : X \longrightarrow Y$, then

$$\mathcal{P}f : \mathcal{P}X \longrightarrow \mathcal{P}Y :: S \mapsto \{f(x) \,:\, x \in S\}.$$

We can compose $\mathcal{P}$ with $\mathcal{T}$ to obtain another presheaf

$$\mathcal{R} := \mathcal{P} \circ \mathcal{T} : \mathbf{Att}^{\mathsf{op}} \longrightarrow \mathbf{Set}.$$

This presheaf assigns the set of A-relations to each set of attributes A; while the restriction map

$$\rho^B_A : \mathcal{R}(B) \longrightarrow \mathcal{R}(A)$$

is exactly the operation of relation restriction, equivalent to the standard notion of projection in relation algebra, which we defined previously:

$$\rho^B_A : R \mapsto R|_A.$$

Natural Join. One of the most important operations in relational algebra is *natural join*. Given an A-relation R and a B-relation S, we define an $(A \cup B)$-relation $R \bowtie S$:

$$R \bowtie S \;:=\; \{t \in \prod_{a \in A \cup B} D_a \,:\, t|_A \in R \;\wedge\; t|_B \in S\}.$$

We shall now show how this operation can be characterized in categorical terms.

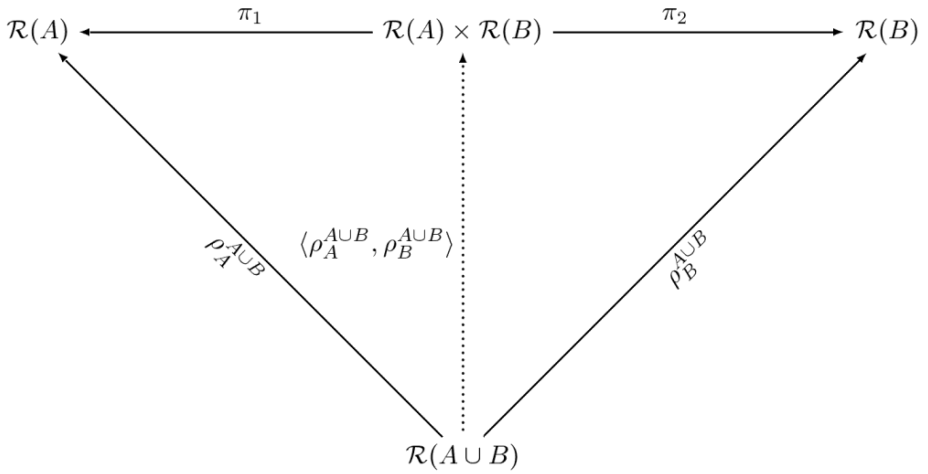
Note firstly that since the powerset is naturally ordered by set inclusion, we can consider $\mathcal{R}$ as a functor

$$\mathcal{R} : \mathbf{Att}^{\mathsf{op}} \longrightarrow \mathbf{Pos}$$

where **Pos** is the category of posets and monotone maps. **Pos** is order-enriched; given monotone maps $f, g : P \to Q$, we can define the pointwise order:

$$f \leq g \;\equiv\; \forall x \in P.\, f(x) \leq g(x).$$

Now suppose we are given attribute sets A and B. We consider the following diagram arising from the universal property of product in **Set**.



Proposition 1. *The natural join* $\bowtie$: $\mathcal{R}(A) \times \mathcal{R}(B) \to \mathcal{R}(A \cup B)$ *is uniquely characterized as the left adjoint of* $\langle \rho_A^{A\cup B}, \rho_B^{A\cup B} \rangle$*; that is, as the unique map satisfying*

$$\mathsf{id}_{\mathcal{R}(A\cup B)} \;\leq\; \bowtie \circ \langle \rho_A^{A\cup B}, \rho_B^{A\cup B} \rangle, \qquad \langle \rho_A^{A\cup B}, \rho_B^{A\cup B} \rangle \circ \bowtie \;\leq\; \mathsf{id}_{\mathcal{R}(A)\times\mathcal{R}(B)}.$$

The fact that in general a relation $R \in \mathcal{R}(A \cup B)$ satisfies only

$$R \;\subseteq\; R|_A \bowtie R|_B,$$

with strict inclusion possible, corresponds to the fact that natural join is in general a 'lossy' operation. Lossless joins correspond exactly to the case when equality holds.

2.3 The Sheaf-Theoretic View

We shall now show, building on the presheaf structure described in the previous sub-section, how a number of important database notions can be interpreted geometrically, in the language of sheaves and presheaves.

Schemas as Covers and Gluing Conditions. We shall interpret a schema $\Sigma = \{A_1, \ldots, A_k\}$ of finite subsets of $\mathcal{A}$ as a *cover*. That is, we think of the attribute sets A_i as 'open sets' expressing some local information in the sense of related clusters of attributes; these sets cover $A := \bigcup_{i=1}^{k} A_i$, the global set of attributes for the schema. Conversely, we think of the global set A as being decomposed into the local clusters A_i; which is exactly the standard point of view in databases.

The basic idea of sheaf theory is to analyze the passage from local to global behaviour in mathematical structures. A number of important notions in databases have exactly this character, and can be described naturally in sheaf-theoretic terms.

An instance $(R_1, \ldots, R_k)$ of a schema Σ is given by specifying a relation $R_i \in \mathcal{R}(A_i)$ for each $A_i \in \Sigma$. In sheaf-theoretic language, this is a family of local sections, defined over the open sets in the cover. A central issue in geometric terms is whether we can glue these local sections together into a global section defined over $A := \bigcup_{i=1}^{k} A_i$.

More precisely, we can ask:

Does there exist a relation $R \in \mathcal{R}(A)$ such that $R|_{A_i} = R_i$, $i = 1, \ldots, k$.

We say that the *gluing condition* is satisfied for the instance $(R_1, \ldots, R_k)$ if such a relation exists.

This has been studied as an algorithmic question in database theory, where it is referred to as the *join consistency property*; it is shown in [17] that it is NP-complete.

Note that a *necessary condition* for this to hold is that, for all $i, j = 1, \ldots, k$:

$$R_i|_{A_i \cap A_j} = R_j|_{A_i \cap A_j}. \tag{1}$$

Indeed, if such an R exists, then

$$R_i|_{A_i \cap A_j} = (R|_{A_i})|_{A_i \cap A_j} = R|_{A_i \cap A_j},$$

using the functoriality of restriction, and similarly for $R_j|_{A_i \cap A_j}$.

We shall say that a database instance $(R_1, \ldots, R_k)$ for which this condition (1) holds has consistent projections, and refer to the family of relations in the instance as a *compatible family.*

These notions can be generalized to apply to any presheaf. If the gluing condition can *always* be satisfied, for any cover and any family of compatible elements, and moreover there is a *unique* element which satisfies it, then the presheaf is a sheaf.

It is of course a well-known fact of life in databases, albeit expressed in a different language, that our relational presheaf $\mathcal{R}$ is *not* a sheaf.

In fact, we have the following:

Proposition 2. *An instance $(R_1, \ldots, R_k)$ satisfies the gluing condition if and only if there is a universal relation R for the instance.*

Here we take a universal relation for the instance by definition to be a relation defined on the whole set of attributes from which each of the relations in the instance can be recovered by projection. This notion, and various related ideas, played an important rôle in early developments in relational database theory; see e.g. [22, 12, 19, 21, 26].

Thus the standard notion of universal relation in databases corresponds exactly to the standard notion of solution to the gluing condition in sheaf theory, for the particular case of the relational presheaf $\mathcal{R}$.

It is also standard that a universal relation need not exist in general, and even if it exists, it need not be unique. There is a substantial literature devoted to the issue of finding conditions under which these properties do hold.

There is a simple connection between universal relations and lossless joins.

Proposition 3. *Let* $(R_1, \ldots, R_k)$ *be an instance for the schema* $\Sigma = \{A_1, \ldots, A_k\}$. *Define* $R := \bowtie_{i=1}^{k} R_i$. *Then a universal relation for the instance exists if and only if* $R|_{A_i} = R_i$, $i = 1, \ldots, k$, *and in this case* R *is the largest relation in* $\mathcal{R}(\bigcup_i A_i)$ *satisfying the gluing condition.*

Proof. We note that, if a relation S satisfies $S|_{A_i} = R_i$, $i = 1, \ldots, k$, then $S \subseteq \bowtie_{i=1}^{k} R_i$ by the adjoint property of the natural join. Moreover, since projection is monotone, in this case $R_i \subseteq S|_{A_i} \subseteq (\bowtie_{i=1}^{k} R_i)|_{A_i} \subseteq R_i$. □

There are further categorical aspects of relational databases which it might prove interesting to pursue. In particular, one can define categories of schemas and of instances and their morphisms, and the construction of colimits in these categories may be applicable to issues of data integration. However, we shall not pursue these ideas here. Instead, we will turn to a natural generalization of relational databases which arises rather effortlessly from the formalism we have developed to this point.

3 Algebraic Databases

We begin by revisiting the definition of the relational presheaf $\mathcal{R}$ in terms of the covariant powerset functor $\mathcal{P}$. An alternative presentation of subsets is in terms of *characteristic functions*. That is, we have the familiar isomorphism $\mathcal{P}(X) \cong \mathbf{2}^X$, where $\mathbf{2} := \{0, 1\}$ is the 2-element boolean algebra.

We can also use this representation to define the functorial action of powerset. Given $s : X \to \mathbf{2}$ and $f : X \to Y$, we define $f^*(s) : Y \to \mathbf{2}$ by

$$f^*(s) : y \mapsto \bigvee_{f(x)=y} s(x). \tag{2}$$

It is easy to see that this is equivalent to

$$f^*(s)(y) = 1 \iff \exists x \in S.\, f(x) = y.$$

Here S is the subset of X whose characteristic function is s.

We can specialise this to the case of an inclusion function $\iota : A \hookrightarrow B$ which induces a map $\mathbf{2}^B \to \mathbf{2}^A$ by restriction:

$$s : B \to \mathbf{2} \quad \mapsto \quad (s|_A) : A \to \mathbf{2}.$$

What we obtain in this case is exactly the notion of *projection* of a relation, as defined in the previous section.

The advantage of this 'matrix' style of definition of the powerset is that it can immediately be generalized rather widely. There is a minor caveat. In the above definition, we used the fact that $\mathbf{2}$ is a *complete* boolean algebra, since there was no restriction on the cardinality of the preimages of f. In the database context, of course, all sets are typically finite.[3] We shall enforce a finiteness condition explicitly in our general definition.

We recall that a *commutative semiring* is a structure $(R, +, 0, \cdot, 1)$, where $(R, +, 0)$ and $(R, \cdot, 1)$ are commutative monoids, and moreover multiplication distributes over addition:

$$x \cdot (y + z) = x \cdot y + x \cdot z.$$

Many examples of commutative semirings arise naturally in Computer Science: we list a few of the most common.

- The reals
$$(\mathbb{R}, +, 0, \times, 1).$$
More generally, any commutative ring is a commutative semiring.
- The non-negative reals
$$(\mathbb{R}_{\geq 0}, +, 0, \times, 1).$$
- The booleans
$$\mathbf{2} = (\{0, 1\}, \vee, 0, \wedge, 1).$$
More generally, *idempotent* commutative semirings are exactly the distributive lattices.
- The min-plus semiring
$$(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, \infty, +, 0).$$

We also note the rôle played by *provenance semirings* in database theory [14,9,11].

We fix a semiring R. Given a set X, the *support* of a function $v : X \to R$ is the set of $x \in X$ such that $v(x) \neq 0$. We write $\mathsf{supp}(v)$ for the support of v. We shall write $\mathcal{V}_R(X)$ for the set of functions $v : X \to R$ of *finite* support. We shall write $\mathcal{D}_R(X)$ for the subset of $\mathcal{V}_R(X)$ of those functions $d : X \to R$ such that

$$\sum_{x \in X} d(x) = 1.$$

Note that the finite support condition ensures that this sum is well-defined.

We shall refer to elements of $\mathcal{V}_R(X)$ as R-valuations on X, and of $\mathcal{D}_R(X)$ as R-distributions.

We consider a few examples:

- If we take $R = \mathbf{2}$, then $\mathcal{V}_R(X)$ is the set of finite subsets of X, and $\mathcal{D}_R(X)$ is the set of finite non-empty subsets.

[3] The sets of data values D_a may be infinite, but only finitely many values will appear in a database instance.

– If we take $R = (\mathbb{R}_{\geq 0}, +, 0, \times, 1)$, then $\mathcal{D}_R(X)$ is the set of discrete (finite-support) probability distributions on X.

Algebraically, $\mathcal{V}_R(X)$ is the free R-semimodule over the set X [13].

These constructions extend to functors on **Set**. Given $f : X \to Y$, we define

$$\mathcal{V}_R(f) : \mathcal{V}_R(X) \to \mathcal{V}_R(Y) :: v \mapsto [y \mapsto \sum_{f(x)=y} v(x)].$$

This restricts to $\mathcal{D}_R$ in a well-defined fashion. Taking $R = \mathbf{2}$, we see that $\mathcal{V}_R(f)$ is exactly the direct image of f, defined as in (2).

We can now generalize databases from the standard relational case to 'relations valued in a semiring' by replacing $\mathcal{P}$ by $\mathcal{V}_R$ (or $\mathcal{D}_R$) in our definition of $\mathcal{R}$; that is, we take $\mathcal{R} := F \circ \mathcal{T}$, where F is $\mathcal{V}_R$ or $\mathcal{D}_R$ for some commutative semiring R. We recover the standard notion exactly when $R = \mathbf{2}$. In the case where $R = (\mathbb{R}_{\geq 0}, +, 0, \times, 1)$ and $F = \mathcal{D}_R$, we obtain a notion of probabilistic database, where each relation specifies a probability distribution over the set of tuples for its attribute-set.

Moreover, our descriptions of the key database operations all generalise to any semiring. If we apply the definition of the functorial action of $\mathcal{V}_R$ or $\mathcal{D}_R$ to the case of restriction maps induced by inclusions, we obtain the right notion of *generalised projection*, which can be applied to any algebraic database. We have already seen that we recover the standard notion of projection in the Boolean case. In the case where the semiring is the non-negative reals, so we are dealing with probability distributions, projection is exactly *marginalization*.

We also note an important connection between probabilistic and relational databases. We can always pass from a probabilistic to a relational instance by taking the *support* of the distribution. Algebraically, this corresponds to mapping all positive probabilities to 1; this is in fact the action of the unique semiring homomorphism from the non-negative reals to the booleans.

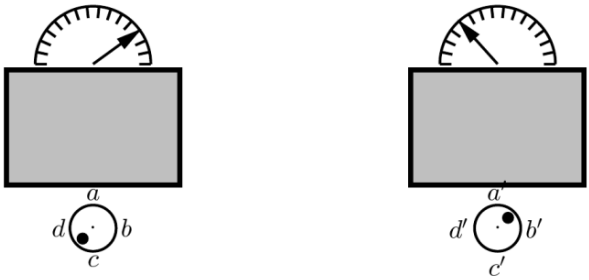
In general, many natural properties of databases will be *preserved* by this homomorphic mapping. This means that if we show that such a property is *not* satisfied by the support, we can conclude that it is not satisfied by the probabilistic instance. Thus we can leverage negative results at the relational level, and lift them to the probabilistic setting.

We shall see a significant example of a probabilistic database in the next section.

4 From Databases to Observational Scenarios

We shall now offer an alternative interpretation of the relational database formalism, with a very different motivation. This will expose a surprising connection between database theory, and on face value a completely different topic, namely Bell's theorem in the foundations of quantum mechanics [8].

Our starting point is the idealized situation depicted in the following diagram.



There are several agents or experimenters, who can each select one of several different measurements $a, b, c, d, \ldots$ to perform, and observe one of several different outcomes. These agents may or may not be spatially separated. When a system is prepared in a certain fashion and measurements are selected, some corresponding outcomes will be observed. These individual occurrences or 'runs' of the system are the basic events. Repeated runs allow relative frequencies to be tabulated, which can be summarized by a probability distribution on events for each selection of measurements. We shall call such a family of probability distributions, one for each choice of measurements, an *empirical model*.

As an example of such a model, consider the following table.

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
$a\ b$	$1/2$	0	0	$1/2$
$a'\ b$	$3/8$	$1/8$	$1/8$	$3/8$
$a\ b'$	$3/8$	$1/8$	$1/8$	$3/8$
$a'\ b'$	$1/8$	$3/8$	$3/8$	$1/8$

The intended scenario here is that Alice can choose between measurement settings a and a', and Bob can choose b or b'. These will correspond to different quantities which can be measured.[4] We assume that these choices are made independently. Thus the *measurement contexts* are

$$\{a, b\}, \quad \{a', b\}, \quad \{a, b'\}, \quad \{a', b'\},$$

and these index the rows of the table. Each measurement has possible outcomes 0 or 1.

Note that, with a small change of perspective, we can see this in database terms. Take the global set of attributes $\mathcal{A} = \{a, a', b, b'\}$, and consider the schema

$$\Sigma := (\{a, b\},\ \{a', b\},\ \{a, b'\},\ \{a', b'\}).$$

For each $a \in \mathcal{A}$, we take $D_a := \{0, 1\}$.

[4] For example, in the quantum case these settings may correspond to different directions along which to measure 'Spin Up' or 'Spin Down' [29].

For each $A \in \Sigma$, we have a 'table' in the algebraically generalized sense discussed in the previous section. That is, we have a distribution $d_A \in \mathcal{D}_R \circ \mathcal{T}(A)$, where $R = \mathbb{R}_{\geq 0}$ is the semiring of non-negative reals. Thus d_A is a probability distribution on $\mathcal{T}(A)$, the set of A-tuples.

To make a direct connection with standard relational databases, we can pass to the support of the above table, which yields the following:

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
$a\ b$	1	0	0	1
$a'\ b$	1	1	1	1
$a\ b'$	1	1	1	1
$a'\ b'$	1	1	1	1

This corresponds to the instance of the schema Σ where for each $A = \{\alpha, \beta\} \in \Sigma \setminus \{\{a, b\}\}$, there is the 'full' table of all possible tuples:

α	β
0	0
0	1
1	0
1	1

while for $\{a, b\}$ we have the table with only two tuples:

a	b
0	0
1	1

Thus we have a formal passage between empirical models and relational databases. To go further, we must understand how empirical models such as these can be used to draw striking conclusions about the foundations of physics.

5 Empirical Models and Hidden Variables

Most of our discussion is independent of any particular physical theory. However, it is important to understand how quantum mechanics, as our most highly confirmed theory, gives rise to a class of empirical models of the kind we have been discussing.

To obtain such a model, we must provide the following ingredients:

- A quantum state.
- For each of the 'measurement settings', which correspond to attributes in database terms, a physical observable or measurable quantity. Each such observable will have a set of associated possible outcomes, which will correspond to the set of data values associated with that attribute.

The 'statistical algorithm' of quantum mechanics will then prescribe a probability for each measurement outcome when the given state is measured with that observable.

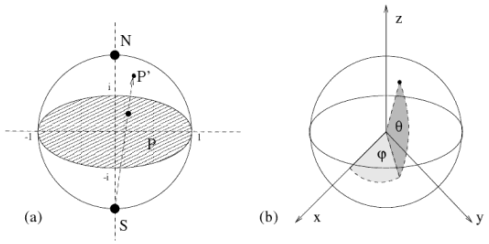
Although we shall not really need the details of this, we briefly recall some basic definitions. For further details, see e.g. [24,29].

A Crash Course in Qubits

Whereas a classical bit register has possible states 0 or 1, a qubit state is given by a *superposition* of these states. More precisely, a (pure) qubit state is given by a vector in the 2-dimensional complex vector space $\mathbb{C}^2$, *i.e.* a complex linear combination $\alpha_0|0\rangle + \alpha_1|1\rangle$, subject to the normalization constraint $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Here $|0\rangle, |1\rangle$ is standard Dirac notation for the basis vectors $[1,0]^T$ and $[0,1]^T$.

Measurement of such a state (in the $|0\rangle$, $|1\rangle$ basis) is inherently probabilistic; we get $|i\rangle$ with probability $|\alpha_i|^2$.

There is a beautiful geometric picture of this complex 2-dimensional geometry in real three-dimensional space. This is the Bloch sphere representation:

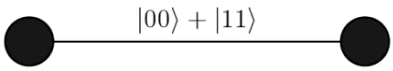


The pure qubit states correspond to points on the surface of the sphere. However, this one-qubit case does not yet provide non-classical resources for information processing. Things get interesting with n-qubit registers

$$\sum_i \alpha_i |i\rangle, \qquad i \in \{0,1\}^n.$$

It is at this point, in particular, that *entanglement phenomena* arise.

A typical example of an entangled state is the Bell state:



We can think of two particles, each with a qubit state, held by Alice and Bob. However, these two particles are entangled. If Alice measures her qubit, then if she gets the answer $|0\rangle$, the state will collapse to $|00\rangle$, and if Bob measures his qubit, he will get the answer $|0\rangle$ with certainty; similarly if the result of Alice's measurement is $|1\rangle$. This non-local effect creates new possibilities for quantum information processing.

Mathematically, compound systems are represented by the *tensor product*, $\mathcal{H}_1 \otimes \mathcal{H}_2$, with typical element

$$\sum_i \lambda_i \cdot \phi_i \otimes \psi_i.$$

Superposition encodes *correlation.*

Entanglement is the physical phenomenon underlying Einstein's 'spooky action at a distance'. Even if the particles are spatially separated, measuring one has an effect on the state of the other.

Bell's achievement was to turn this puzzling feature of quantum mechanics into a theorem: quantum mechanics is *essentially non-local*.

5.1 Bell's Theorem

We look again at the empirical model

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
(a,b)	$1/2$	0	0	$1/2$
(a,b')	$3/8$	$1/8$	$1/8$	$3/8$
(a',b)	$3/8$	$1/8$	$1/8$	$3/8$
(a',b')	$1/8$	$3/8$	$3/8$	$1/8$

This can be realized in quantum mechanics, using a Bell state

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}},$$

subjected to measurements in the XY-plane of the Bloch sphere, at relative angle $\pi/3$. Systems of this kind have been the subject of extensive experimental investigation, and the predictions of quantum mechanics can be taken to be very highly confirmed.

The question we shall ask, following Bell, is this: Can we explain these empirical findings by a theory which is *local* and *realistic* in the following sense.

- A theory is realistic if it ascribes definite values to all observables for every physical state, independently of the activities of any external observers.
- A theory is local if the outcomes of measurements on spatially separated subsystems depend only on common causal factors. In particular, for space-like separated measurements, the outcomes of the measurements should be independent of each other.

We allow for the fact that there may be salient features in the theory determining the outcomes of measurements of which we are not aware. These features are embodied in the notion of *hidden variable*. Thus we take measurement outcomes to be determined, *given some value of this hidden variable*. Moreover, we assume that this hidden variable acts in a local fashion with respect to spatially separated subsystems.

This gives a general notion of theory which behaves in a fashion broadly consistent with classical physical intuitions. The import of Bell's theorem is exactly that *no such theory can account for the empirical predictions of quantum mechanics*. Hence, given that these predictions are so well-confirmed, we must abandon the classical world-view which underpins the assumptions of local realism.

To give a precise statement of Bell's theorem, we must formalize the notion of local hidden variable theory. We shall give this in a streamlined form, which can be shown to be equivalent to more general definitions which have been considered (see e.g. Theorem 7.1 in [3]).

We shall explain this notion in relation to the Bell table given above. We have a total set of four measurement settings we are considering, two for Alice and two for Bob:

$$\{a, a', b, b'\}.$$

A simultaneous assignment of outcomes (0 or 1) to each of these is given by a function

$$s : \{a, a', b, b'\} \longrightarrow \{0, 1\}.$$

The fact that an (unknown) hidden variable may be affecting the outcome is captured by saying that we have a probability distribution d on the set of all such functions s. Such a probability distribution can be taken to be a canonical form for a hidden variable.

The requirement on this distribution d to be consistent with the empirical data is that, for each of the experimentally accessible combinations of measurement settings

$$\{a, b\}, \quad \{a', b\}, \quad \{a, b'\}, \quad \{a', b'\},$$

the restriction (or marginalization) of d to this set of measurements yields exactly the observed distribution on outcomes from the corresponding row of the table. For example, we must have $d|\{a, b\} = d_1$, where

$$d_1(0, 0) = d_1(1, 1) = 1/2, \quad d_1(0, 1) = d_1(1, 0) = 0.$$

A precise statement of a particular instance of Bell's theorem can now be given as follows:

Proposition 4. *There is no distribution d on the whole set of measurements which yields the observable distributions by restriction.*

Proof. Assume for a contradiction that such a distribution d exists. It will assign a number $X_i \in [0, 1]$ to each $s_i : \{a, a', b, b'\} \longrightarrow \{0, 1\}$. There are 16 such functions: we enumerate them by viewing them as binary strings, where the j'th bit indicates the assignment of an outcome to the j'th measurement, listed as a, a', b, b'.

The requirement that this distribution projects onto the distributions in the empirical model translates into 16 equations, one for each entry in the table. It suffices to consider 4 of these equations:

$$\begin{aligned} X_1 + X_2 + X_3 \ + X_4 \ &= 1/2 \\ X_2 + X_4 + X_6 \ + X_8 \ &= 1/8 \\ X_3 + X_4 + X_{11} + X_{12} &= 1/8 \\ X_1 + X_5 + X_9 \ + X_{13} &= 1/8 \end{aligned}$$

Adding the last three equations yields

$$X_1 + X_2 + X_3 + 2X_4 + X_5 + X_6 + X_8 + X_9 + X_{11} + X_{12} + X_{13} \ = \ 3/8.$$

Since all these terms must be non-negative, the left-hand side of this equation must be greater than or equal to the left-hand side of the first equation, yielding the required contradiction. □

This argument seems very specific to the probabilistic nature of the empirical model. However, an important theme in the work on no-go theorems in quantum mechanics is to prove results of this kind in a probability-free fashion [15,16]. This will bring us directly into the arena of relational databases.

5.2 Hardy's Construction

Hardy's construction [16] yields a family of empirical models which can be realized in quantum mechanics in similar fashion to the Bell model. However, these families exhibit a stronger form of non-locality property, which does not depend on the probabilities, but only on the support.

We exhibit an example of a support table arising from Hardy's construction.

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
(a,b)	1	1	1	1
(a',b)	0	1	1	1
(a,b')	0	1	1	1
(a',b')	1	1	1	0

This arises from a probability table by replacing all positive probabilites by 1.

Note that we can view this table as encoding a small relational database, as in our discussion in the previous section. There will be four relation tables in this database, one for each of the above rows. The table corresponding to the first row will have the full set of tuples over $\{0,1\}$. The tables for the second and third rows will have the form

α	β
0	1
1	0
1	1

while that for the fourth row will have the form

a'	b'
0	0
0	1
1	0

The property which shows the non-locality of this model is the exact relational analogue of the probabilistic version we considered in relation to the Bell model.

Proposition 5. *There is no A-relation R, where $A = \{a, a', b, b'\}$, which is consistent with the empirical observable supports; that is, for which $R|\{\alpha,\beta\}$ yields the relational table for all $\{\alpha,\beta\}$, $\alpha \in \{a,a'\}$, $\beta \in \{b,b'\}$.*

In database language, this says exactly that there is no 'universal relation' on the whole set of attributes which yields each of the 'observable relations' by projection.

Proof. We consider the case where $n = 4k$, $k \geq 1$. Assume for a contradiction that we have a global section $s \in S_e$ for the GHZ model e.

If we take Y measurements at every part, the number of 1 outcomes under the assignment is even. Replacing any two Y's by X's changes the residue class mod 4 of the number of Y's, and hence must result in the opposite parity for the number of 1 outcomes under the assignment. Thus for any $Y^{(i)}$, $Y^{(j)}$ assigned the *same* value, if we substitute X's in those positions they must receive *different* values under s. Similarly, for any $Y^{(i)}$, $Y^{(j)}$ assigned different values, the corresponding $X^{(i)}$, $X^{(j)}$ must receive the same value.

Suppose firstly that not all $Y^{(i)}$ are assigned the same value by s. Then for some i, j, k, $Y^{(i)}$ is assigned the same value as $Y^{(j)}$, and $Y^{(j)}$ is assigned a different value to $Y^{(k)}$. Thus $Y^{(i)}$ is also assigned a different value to $Y^{(k)}$. Then $X^{(i)}$ is assigned the same value as $X^{(k)}$, and $X^{(j)}$ is assigned the same value as $X^{(k)}$. By transitivity, $X^{(i)}$ is assigned the same value as $X^{(j)}$, yielding a contradiction.

The remaining cases are where all Y's receive the same value. Then any pair of X's must receive different values. But taking any 3 X's, this yields a contradiction, since there are only two values, so some pair must receive the same value.

The case when $n = 4k+2$, $k \geq 1$, is proved in the same fashion, interchanging the parities. When $n \geq 5$ is odd, we start with a context containing one X, and again proceed similarly.

The most familiar case, for $n = 3$, does not admit this argument, which relies on having at least 4 Y's in the initial configuration. However, for this case one can easily adapt the well-known argument of Mermin using 'instruction sets' [23] to prove strong contextuality. This uses a case analysis to show that there are 8 possible global sections satisfying the parity constraint on the 3 measurement combinations with 2 Y's and 1 X; and all of these violate the constraint for the XXX measurement. □

6.2 The Kochen-Specker Theorem

Kochen-Specker-type theorems [18] can be understood as *generic strong contextuality results*. In database terms, they say that, if the database schema has a certain combinatorial structure, then *every instance* satisfying some conditions is strongly contextual. This can be interpreted in the quantum context in such a way that the conditions will be satisfied by *every* quantum state, and hence we obtain a state-independent form of strong contextuality result.

The condition which is typically imposed on the instances, assuming that the possible data values for each attribute lie in $\{0, 1\}$, is that *every tuple contains exactly one* 1. If we think in terms of satisfiability, this corresponds to a 'POSITIVE ONE-IN-k-SAT' condition.

To show that the Kochen-Specker result holds is exactly to show that there is no satisfying assignment for the corresponding set of clauses.

The simplest example of this situation is the 'triangle', *i.e.* the schema with elements

$$\{a, b\}, \{b, c\}, \{a, c\}.$$

However, this example cannot be realized in quantum mechanics [3].

An example which can be realized in quantum mechanics, where $\mathcal{A}$ has 18 elements, and there are 9 sets in the database schema, each with four elements, such that each element of $\mathcal{A}$ is in two of these, appears in the 18-vector proof of the Kochen-Specker Theorem in [10].

U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9
A	A	H	H	B	I	P	P	Q
B	E	I	K	E	K	Q	R	R
C	F	C	G	M	N	D	F	M
D	G	J	L	N	O	J	L	O

Here the schema is $\Sigma = \{U_1, \ldots, U_9\}$.

We shall give a simple combinatorial condition on the schema Σ which is implied by the existence of a global section s satisfying the 'POSITIVE ONE-IN-k-SAT' condition. Violation of this condition therefore suffices to prove that no such global section exists.

For each $a \in \mathcal{A}$, we define

$$\Sigma(a) := \{A \in \Sigma \,:\, a \in A\}.$$

Proposition 8. *If a global section satisfying the condition exists, then every common divisor of* $\{|\Sigma(a)| \,:\, a \in \mathcal{A}\}$ *must divide* $|\Sigma|$.

Proof. Suppose there is a global section $s : \mathcal{A} \to \{0, 1\}$ satisfying the condition. Consider the set $X \subseteq \mathcal{A}$ of those a such that $s(a) = 1$. Exactly one element of X must occur in every $A \in \Sigma$. Hence there is a partition of Σ into the subsets $\Sigma(a)$ indexed by the elements of X. Thus

$$|\Sigma| = \sum_{a \in X} |\Sigma(a)|.$$

It follows that, if there is a common divisor of the numbers $|\Sigma(a)|$, it must divide $|\Sigma|$. □

For example, if every $a \in \mathcal{A}$ appears in an even number of elements of Σ, while Σ has an odd number of elements, then there is no global section. This corresponds to the 'parity proofs' which are often used in verifying Kochen-Specker-type results [10,28]. For example, in the 18-attribute schema with 9 relations given above, each attribute appears in two relations in the schema; hence the argument applies.

For further discussion of these ideas, including connections with graph theory, see [3].

7 Further Directions

We mention some further directions for developing the connections between databases and the study of non-locality and contextuality in quantum mechanics.

- We may consider conditions on the database schema which guarantees that global sections can be found. The important notion of *acyclicity* in database theory [7] is relevant here. On the probabilistic side there is a result by Vorob'ev [27] (motivated by game theory), which gives necessary and sufficient combinatorial conditions on a schema for *any* assignment of probability distributions on the tuples for each relation in the schema to have a global section; that is, for a universal relation in the probabilistic sense to always exist for any probabilistic instance of the database. Rui Soares Barbosa (personal communication) has shown that the Vorob'ev condition is equivalent to acyclicity in the database sense. This provides another striking connection between database theory and the theory of quantum non-locality and contextuality.
- A logical approach to Bell inequalities in terms of logical consistency conditions is developed in [5]. It would be interesting to interpret and apply this notion of Bell inequalities in the database context.
- The tools of sheaf cohomology are used to characterize the obstructions to global sections in a large family of cases in [6]. In principle, these sophisticated tools can be applied to databases. There may be interesting connections with acyclicity in the database sense.

We can summarise the connections which we have exposed between database theory and quantum non-locality and contextually in the following table:

Relational databases	measurement scenarios
attribute	measurement
set of attributes defining a relation table	compatible set of measurements
database schema	measurement cover
tuple	local section (joint outcome)
relation/set of tuples	boolean distribution on joint outcomes
universal relation instance	global section/hidden variable model
acyclicity	Vorob'ev condition

Acknowledgements. Discussions with and detailed comments by Phokion Kolaitis are gratefully acknowledged. Leonid Libkin also gave valuable feedback. This paper was written while in attendance at the program on 'Semantics and Syntax: the legacy of Alan Turing' at the Isaac Newton Institute, Cambridge, April–May 2012.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Abramsky, S.: Relational Hidden Variables and Non-Locality. Studia Logica 101(2), 411–452 (2013)
3. Abramsky, S., Brandenburger, A.: The sheaf-theoretic structure of non-locality and contextuality. New Journal of Physics 13(2011), 113036 (2011)
4. Abramsky, S., Gottlob, G., Kolaitis, P.: Robust constraint satisfaction and local hidden variables in quantum mechanics. In: Rossi, F. (ed.) Proceedings of the International Joint Conference in Artificial Intelligence (IJCAI) (2013)
5. Abramsky, S., Hardy, L.: Logical Bell Inequalities. Physical Review A 85, 062114 (2012)
6. Abramsky, S., Mansfield, S., Barbosa, R.S.: The cohomology of non-locality and contextuality. In: Proceedings of Quantum Physics and Logic 2011. EPTCS, vol. 95, pp. 1–15 (2012)
7. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of the ACM (JACM) 30(3), 479–513 (1983)
8. Bell, J.S.: On the Einstein-Podolsky-Rosen paradox. Physics 1(3), 195–200 (1964)
9. Buneman, P., Tan, W.C.: Provenance in databases. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 1171–1173. ACM (2007)
10. Cabello, A., Estebaranz, J.M., García-Alcaine, G.: Bell-Kochen-Specker theorem: A proof with 18 vectors. Physics Letters A 212(4), 183–187 (1996)
11. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Foundations and Trends in Databases 1(4), 379–474 (2009)
12. Fagin, R., Mendelzon, A.O., Ullman, J.D.: A simplified universal relation assumption and its properties. ACM Transactions on Database Systems (TODS) 7(3), 343–360 (1982)
13. Golan, J.S.: Semirings and their Applications. Springer (1999)
14. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 31–40. ACM (2007)
15. Greenberger, D.M., Horne, M.A., Zeilinger, A.: Going beyond Bell's theorem. In: Kafatos, M. (ed.) Bell's Theorem, Quantum Theory, and Conceptions of the Universe, pp. 69–72. Kluwer (1989)
16. Hardy, L.: Quantum mechanics, local realistic theories, and Lorentz-invariant realistic theories. Physical Review Letters 68(20), 2981–2984 (1992)
17. Honeyman, P., Ladner, R.E., Yannakakis, M.: Testing the universal instance assumption. Information Processing Letters 10(1), 14–19 (1980)
18. Kochen, S., Specker, E.P.: The problem of hidden variables in quantum mechanics. Journal of Mathematics and Mechanics 17(1), 59–87 (1967)
19. Korth, H.F., Kuper, G.M., Feigenbaum, J., Van Gelder, A., Ullman, J.D.: SYSTEM/U: A database system based on the universal relation assumption. ACM Transactions on Database Systems (TODS) 9(3), 331–347 (1984)
20. Mac Lane, S.: Categories for the working mathematician, vol. 5. Springer (1998)
21. Maier, D., Ullman, J.D.: Maximal objects and the semantics of universal relation databases. ACM Transactions on Database Systems (TODS) 8(1), 1–14 (1983)
22. Maier, D., Ullman, J.D., Vardi, M.Y.: On the foundations of the universal relation model. ACM Transactions on Database Systems (TODS) 9(2), 283–308 (1984)

23. Mermin, N.D.: Quantum mysteries revisited. Am. J. Phys. 58(8), 731–734 (1990)
24. Nielsen, M.Q.C., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
25. Pierce, B.C.: Basic category theory for computer scientists. The MIT Press (1991)
26. Ullman, J.D.: Principles of database systems. Prentice-Hall (1983)
27. Vorob'ev, N.N.: Consistent families of measures and their extensions. Theory of Probability and its Applications 7, 147 (1962)
28. Waegell, M., Aravind, P.K.: Parity proofs of the Kochen-Specker theorem based on the 24 rays of Peres. Arxiv preprint arXiv:1103.6058v1 (2011)
29. Yanofsky, N.S., Mannucci, M.A.: Quantum computing for computer scientists, vol. 20. Cambridge University Press, Cambridge (2008)

language design choices and primitives of HIL, as well as its compilation and execution, are described in [22]. While HIL answers some of the research challenges outlined in this article, several important problems remain largely open, such as the need for tools or systems to support large-scale data exploration or to assist users with the actual development of a *good* set of data analysis rules.

1.1 Overview of the Paper

We start in Section 2 by describing some of the features of the data in DBpedia, as well as the challenges involved in data exploration, which is a phase that *precedes* the actual writing of the rules. We then illustrate some concrete rules for extracting facts from DBpedia. Here, the output of an extraction rule has a relatively simple structure (or schema), but the input is semi-structured and largely heterogeneous. Extraction from completely unstructured data (i.e., text) [11] is highly related in this context; however, in this paper, we focus our attention specifically on extraction from semi-structured data (e.g., RDF, or XML, or JSON). We also note that extraction from text, technically, is of a different nature and is discussed extensively elsewhere (e.g., [8]).

In addition to giving examples of extraction rules, we also include a discussion of the need for automatic or semi-automatic extraction of structured records that is based on data examples. Such technology, while non-trivial, would be particularly useful when the developer is in the exploration phase and does not know enough about the data and its peculiarities. Based on a few examples that are representative of the type of entities that the developer is interested to extract, the system must first be able to derive all the other entries that are "similar" to the given examples. More challenging, the system should come up with a set of extraction rules that would result in such entries. While existing work on query discovery based on data instances [18,27] or on schema mapping design based on examples [1,21] may provide a starting point here, new types of algorithms will have to be developed to account for highly heterogeneous data with "less" schema (such as DBpedia).

The next integration component that we address is entity resolution, in Section 3. Rather than looking at specific algorithms or implementations that match records based on various similarity measures on their fields, we take a higher-level approach where the goal is to provide the *specification* framework for entity resolution. We advocate a framework that is based on logical constraints that are similar, in spirit, to the dependencies used in data exchange [15]. However, different from data exchange where the dependencies are source-to-target, our entity resolution constraints are target-to-source: they define *declaratively* all the desired properties of the target (i.e., of the links) in terms of the sources. Furthermore, these constraints incorporate disjunction (of the alternative matching rules that may apply), rely on user-defined functions for computing similarity of values, and can include cardinality constraints (e.g., to express many-to-one type of links). We include a discussion to illustrate the differences between this framework and previous approaches such as the Dedupalog language [2].

One of the main research problems that we outline, as part of declarative entity resolution, is the compilation of the declarative constraints into an execution plan that produces a good instantiation of the links. An important related question is formulating the semantics of the declarative constraints, which then needs to be implemented by the execution plan. Finally, a major challenge for entity resolution, which goes beyond the design of the specification language, is the development of methods and tools to help users interactively resolve the inherent ambiguities in their specification. These tools can help users refine the declarative constraints, based on the actual data sets that need to be linked, to ultimately achieve a high quality specification for entity resolution.

We discuss mapping and transformation, as well as data fusion and aggregation aspects in Section 4. While there is work on schema mapping tools [14], data exchange semantics [15], and data fusion methods [6], our goal is to develop an expressive scripting language that allows developers to combine non-trivial mapping, fusion and aggregation tasks (e.g., that are often not possible within a schema mapping tool paradigm) with the declarative entity resolution and extraction operations discussed earlier. At the same time, we emphasize simplicity and ease of programming as important requirements for the language design.

We discuss several other related papers and systems in Section 5 and conclude the paper in Section 6, where we reiterate the need for a single, unified framework that incorporates all the aspects outlined in the previous sections.

2 Data Exploration and Extraction

The first step before the actual writing of extraction and integration rules is the *exploration* phase, where a human user needs to understand what is in the source data and what can be extracted. This step is usually expensive; any help that a system or tool can provide in assisting the human user can be valuable. Even if the user has an idea of what concepts need to be extracted, the form in which these concepts manifest in the actual data source can vary significantly. Hence, heterogeneity is a challenge.

We start with an example from DBpedia to illustrate the issues. We focus on financial companies (e.g., Bank of America, Citigroup); the goal here will be to extract structured records that are relevant for such financial companies and that are deemed useful towards building the final integrated view. First, we assume that the DBpedia data set is given as a set of JSON records, each corresponding to one entity. A record has a subject field (which is also the identifier of that entity), and then all the various properties recorded for that entity. This JSON representation can be easily obtained from the RDF version of Dbpedia, which records RDF triples of the form (subject, property, value).[2] The conversion from RDF to JSON is already a step towards a more unified view of the data, since it yields full objects rather triples. However, the format of these objects is wildly heterogeneous, even for the same "type" of entity, as we shall see shortly. A large

[2] See the Ontology Infobox Properties data set at `http://wiki.dbpedia.org/Downloads`.

```
{
  "assets": “US$ 2.264 trillion",
  "foundation": “1904",
  "homepage": [ "http://www.bankofamerica.com",
                “http://www.bofa.com" ],
  "industry": [ "Banking", “Financial services" ]
  "keyPeople": [
   “Bryan Moynihan",
   “(President and CEO)",
   “Charles Holliday",
   “(Chairman)"
  ],
  "location": [
   "Charlotte,_North_Carolina",
   "United_States",
   "North_Carolina"
  ],
  "name": "Bank of America Corporation",
  "numEmployees": "288000",
  "slogan": "Bank of Opportunity",
  "subject": "Bank_of_America",
  "type": "Public_company",
  "wikiPageUsesTemplate": "Template:infobox_company"
},
```

```
{
  "areaServed": "Worldwide",
  "assets": "$ 1.119 trillion (2007)",
  "companyName": "Goldman_Sachs",
  "companySlogan": "Our clients\' interests always come first",
  "companyType": "Public_company",
  "foundation": "1869",
  "founder": ["Marcus_Goldman“, “Samuel Sachs"],
  "homepage": "http://www.gs.com/",
  "industry": "Finance_and_insurance",
  "keyPeople": [
   "Lloyd_Blankfein",
   (Chairman & CEO)",
   "Gary_Cohn",
   “(President & COO)",
   “David Viniar“,
   “(Executive VP & CFO)"
  ],
  "location": [ "United_States", "New_York_City" ],
  "marketCap": "$ 65.91 billion (2007)",
  "numEmployees": "30,522 (2007)",
  "products": [
   "Financial_services",
   "Investment_bank"
  ],
  "revenue": "$ 87.968 billion (2007)",
  "subject": "Goldman_Sachs",
  "wikiPageUsesTemplate": "Template:infobox_company"
},
```


Fig. 1. Sample DBpedia records

part of the subsequent processing will be devoted to extracting the relevant parts of the objects of interest, bringing the extracted parts to a uniform format, and then linking and integrating them with data from other sources (e.g., SEC).

Figure 1 illustrates two sample input records, in JSON, corresponding to the DBpedia entries for Bank of America and Goldman Sachs. Even though both of these records represent entities of a similar type (i.e., financial institutions), there is significant variation in the structure of the records (i.e., the attributes that are present, their types), in the naming of the attributes, and in the values and format of the values that populate the attributes. For example, Goldman Sachs has attributes such as “founder” and “marketCap”, while Bank of America does not include these attributes. Goldman Sachs has a “companyName” attribute, while the equivalent attribute for Bank of America is “name”. The “homepage” attribute for Goldman Sachs is a single string, while the similar attribute for Bank of America is an array of strings. Finally, the values themselves are not always clean or cleanly organized. For example, Bank of America includes “Banking” and “Financial services” under the “industry” attribute; the corresponding information for Goldman Sachs is actually distributed over two attributes (“industry” and “products”). Furthermore, the entries under the “keyPeople” attribute, in both records, are a mixture of person names and positions (titles), without an explicit tagging of the data.

After exploring several more representative DBPedia entries for financial companies, the user may decide on a set of important *concepts* to be extracted from this collection of heterogeneous records. Each concept is based on a subset of

```
FinancialCompany =
    for (r in DBpedia)
    let industryTerms = extractIndustries(r.industry),
        compName = extractCompanyName(r)
    where contains (compName, "Bank|Insurance|Investment") or
          (some (i in industryTerms) satisfies
              contains (i, "bank|banking|insurance|finance|financial"))
    return {company_id: r.subject,
            name: compName,
            foundation: r.foundation,
            industry: industryTerms,
            revenue: cleanDollarAmount(r.revenue)
           }
```

Fig. 2. Extraction rule for financial companies

attributes and, hence, it is a piece of a schema. In our scenario, the user may be interested in the following three concepts.

FinancialCompany (company_id, name, foundation, industry, revenue, ...)
CompanyAddress (company_id, street1, street2, zipcode, city, state, country)
KeyPeople (person_name, titles, company_name, age, biography, ...)

Note that, in general, the schema for these concepts must be *open* (see the above ... notation) to account for possibly other attributes of interest that may be added later. The high-level integration language will have to be flexible and account for such open schema by either not requiring the user to explicitly having to define the schemas of the concepts, or by using advanced programming language features such as record polymorphism to represent extensible record types [24,25,28].

Finally, other concepts can be defined later from either the same source (DBPedia) or from other sources (e.g., SEC, as we will see later). All of these extracted concepts will then be processed together, in the subsequent integration flow, to generate clean target entities with richer structure.

We focus next on how to extract the data to populate such concepts from the underlying collection of heterogeneous records.

2.1 Extraction Rules: Examples

Figure 2 gives a first example of a rule that extracts data for financial companies from DBpedia. This rule populates into the FinancialCompany concept. There may be other rules to further populate into this same concept (and possibly add new attributes). Thus, the actual instance of a concept will be given by a union of extraction rules.

The rule uses an XQuery-like syntax (although other types of syntax could also be used) to express the search for DBPedia records that match the characteristics of a financial company and also to express the extraction of the relevant

attributes. Note the complex predicate that is used in the where clause to recognize a financial company. This predicate includes multiple string matching conditions that are based on financial keywords. Note also the extensive presence of user-defined functions (UDFs) that are used for various purposes:

- to *clean* the data in the individual attributes. For example, cleanDollarAmount is a function that transforms various heterogeneous string values that represent dollar amounts into a standardized form. Concretely, strings such as "$ 87.968 billion (2007)" and "US$ 2.264 trillion" could be transformed into "$87.96 billion" and "$2.26 trillion", respectively.
- to *extract* certain expected strings from an input record or value (e.g., extractCompanyName from r and extractIndustries from r.industry).
- more generally, to account for the heterogeneity in the input data or structure. For example, extractIndustries must account for the fact that the input r.industry could be a string such as "Finance_and_insurance" or an array such as ["Banking", "Financial services"]. The function must uniformly generate an array of terms identifying the various relevant industries (i.e., ["finance", "insurance"] from the first input and ["banking", "financial services"] from the second input).

  As another example, extractCompanyName has to account for the fact that the company name can appear under various attributes in the input record r (e.g., sometime name, and sometime companyName). Furthermore, the value itself must be normalized (e.g., "Goldman_Sachs" must be transformed to "Goldman Sachs").

  Note that the extracted and normalized industry terms and company name are used both in the predicate in the where clause that identifies a financial company and in the output of the rule.

In Figure 3, we show another example of an extraction rule from DBPedia, to produce records for the key people that are associated with the financial companies. As before, the rule makes use of UDFs to restrict to financial companies. An additional UDF extractNameTitles is used to convert an array of strings into a set of structured records with explicit name and titles fields. For example, the array of uninterpreted strings that is the value of the keyPeople field in the "Goldman Sachs" record in Figure 1 is converted into a set of three records:

```
{ name: "Lloyd Blankfein", titles: ["Chairman", "CEO"] }
{ name: "Gary Cohn", titles: ["President", "CEO"] }
{ name: "David Viniar", titles: ["Executive VP", "CFO"] }
```

Note that the above UDF must employ a name recognizer as well as a title recognizer. Also, it must take into account the sequence in which the names and the titles appear in the input string. In particular, the function must detect that the titles of a person follow the actual person name, and also it must be able to handle the absence of title information (e.g., two consecutive names).

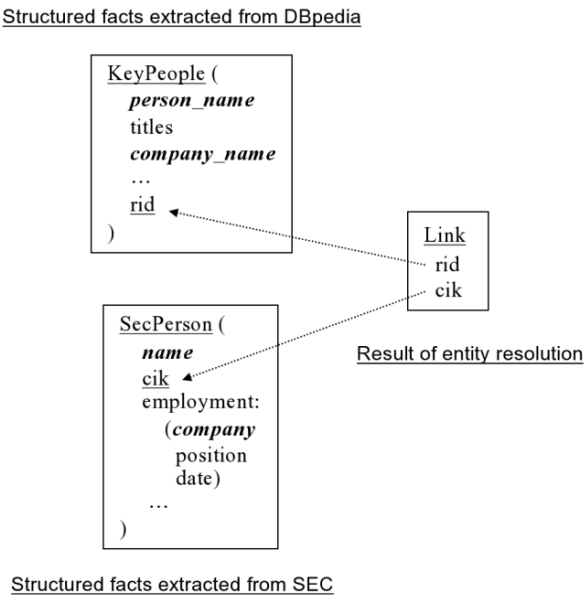


Fig. 4. Entity resolution diagram

other contextual information such as employment. Figure 4 depicts schematically the concrete entity resolution scenario that we are considering.

3.1 Declarative Constraints for Entity Resolution

We now illustrate the logic that is needed to express the above entity resolution problem. We advocate a declarative formalism where one specifies the properties or *constraints* that the outcome of entity resolution (i.e., the link table) must satisfy, without having to specify a concrete procedure or implementation for computing this outcome. It will be the role of the underlying system to materialize a good solution (i.e., a set of links) that satisfies the specified constraints in the best possible way.

For our entity resolution example, we show in Figure 5 a set of declarative constraints that can be used to specify the desired properties of the link table. We believe that such constraints (and their extensions) should form the basic ingredients of any language that attempts to specify entity resolution at a high-level.[3] We explain the constraints first and then discuss the issues involved in building a language and system that implements such specification.

First, we have provenance or identification constraints that specify the attributes or combinations of attributes that identify the source objects to be

[3] However, the syntax of the actual language does not have to have follow the logical notation we use here. Furthermore, some of these constraints may be implicit in the semantics of the language.

```
Link [rid] ⊆ KeyPeople [rid]
Link [cik] ⊆ SecPerson [cik]

Link : rid → cik

(m) every Link
    satisfies
        KeyPeople.person_name = SecPerson.name
    or
      (KeyPeople. person_name ~name SecPerson.name
       and
       KeyPeople.company_name in SecPerson.employment [company]
      )
```

Fig. 5. Declarative constraints for entity resolution

linked. In this example, the two inclusion dependencies from Link to the sources specify that the projection of Link on rid must be a subset of the projection of KeyPeople on rid and, similarly, the projection of Link on cik must be a subset of the projection of SecPerson on cik. Thus, the intention behind Link is to be a subset of all the pairs of rid and cik values that appear in the two sources. In general, it is up to the user to define what constitutes the identifier of an object of interest for entity resolution. The framework we suggest is independent of what makes the identifier of an object. As a result, we can naturally capture most types of entity resolution described in the literature, from record linkage and deduplication [17,23] to reference reconciliation [12] and to more general, semantic type of linkage among entities (e.g., the relationship between companies and subsidiaries). To follow some of the terminology in the literature, in our example, the first type of object that participates in Link can be viewed as an entity reference (since it refers indirectly to an actual person, via person name and other non-identifying attributes), while the second type of object can be viewed as an entity (since it identifies a person in SEC).

The next constraint in the specification is a functional dependency (on the Link table) to specify that an rid from the first source must be linked to a unique cik in the second source. Note that, in this example, it is is ok to have multiple rid's linked to the same person cik. Thus, by using a functional dependency, we encode an N:1 type of entity resolution (where multiple objects of interest in one source must be linked to a single object in another source). For 1:1 type of entity resolution, we would write a functional dependency in the other direction as well. For an N:M type of entity resolution, we do not need to specify any functional dependencies.

The final constraint in this example, probably the most important, is used to declare a disjunction of all the valid reasons for why two objects can match. Essentially this constraint specifies that a link can exist only if at least one of several matching conditions holds. The matching conditions are formulated with

respect to the source tuples that are related via the link. In the example, we can have a match because of exact equality of person names, or because of similarity of person names (via a user-defined similarity predicate) and, moreover, because the company_name in the KeyPeople record appears in the employer set in the SecPerson record. Note that the second matching condition relaxes the equality on person names, when compared to the first matching rule, but at the same adds a strenghtening condition that is based on employment information. Note that the employment-based condition, although a strengthening, may apply to less tuples (those that have a non-empty employment set in SecPerson). In practice, one will have to formulate multiple matching conditions, in order to improve the recall of entity resolution. Furthermore, each matching condition has to be strong enough to prevent the generation of accidental links.

Other types of constraints that appear in practice are structural type of constraints requiring properties such as transitivity of matching or variations of it. Such constraints are needed to specify clustering behavior or to specify the linking of two objects in two sources due to another object in a third source that links to them.

A slight extension to this basic framework of constraints allows us to express *collective entity resolution* [5], where the task is to create multiple, inter-related types of links (rather than to create a single type of link). For example, assume that we have the following two source relations:

```
Paper (pid, title, venue, year, ...)
Venue (venue, conferenceOrJournal, sponsor, ...)
```

In this context, we may want to specify links between papers *and* links between venues. Assume that the first type of link is represented as a binary relation PaperLink(pid1, pid2), while the second type of link is represented as a binary relation VenueLink(venue1, venue2). Then, the matching rules for one type of link may depend on the other type of link. For example, we can declare the matching conditions for VenueLink as follows:

```
every VenueLink satisfies
   ... (some similarity condition on venue names) ...
or
   ... (other condition) ...
or
   exists (p1 in Paper, p2 in Paper)
      p1.venue = VenueLink.venue1 and p2.venue = VenueLink.venue2 and
      PaperLink (p1.pid, p2.pid)
```

In particular, the last condition says that a possible reason for a venue link is that there exist two papers that are linked via PaperLink and whose venues are the two venues related by the link.

Note that in the framework we suggest, we do not *force* the generation of links, but rather define them *implicitly* through a declaration of the possible matching rules. For example, satisfying the last matching condition in the above

constraint does not mean that a VenueLink tuple will necessarily be created, since the existence of such tuple may be prevented due to other constraints. In fact, creating such link may be the wrong choice sometimes (e.g., a conference version and a journal version of a paper may be linked via PaperLink, but that does not mean that the conference and the journal represent the same venue). The disjunction allows us to enumerate, declaratively, all the possible reasons for why a link may exist without forcing the link generation. It is then the job of the underlying system to take into account all the constraints to reach a good set of links, as we discuss in the next section.

Other frameworks aimed at declarative entity resolution exist. Perhaps, the most comprehensive one is the Dedupalog [2] language which allows the use of constraints, expressed in a Datalog style of syntax, to drive the identification of duplicate entities. Several remarks are in order here. First, Dedupalog limits itself to links that are equivalence relations, thus focusing strictly on deduplication. In contrast, we require a more flexible framework for links that represent more general semantic relationships, going beyond the "same-as" type of relationship. Furthermore, Dedupalog rules are not entirely declarative. Generally speaking, rules in Dedupalog are a guideline for the implementation, and the intention of a rule is to populate links based on conditions on the sources or other links. Since forcing links may create inconsistencies in the result, Dedupalog compensates by allowing some rules to be soft: for such rules, links are "likely" to be generated. The system then figures out to what extent to satisfy these rules (e.g., by attempting to minimize the overall number of constraint violations). As a consequence, an important downside is that the result of Dedupalog evaluation does not satisfy, in a precise first-order logic sense, the Dedupalog rules that were given as a specification. Furthermore, it may not be easy for a user of the system to understand the properties of the final result.

In contrast, the matching constraints that we envision have a purely declarative flavor, where we specify all the desired properties on the target links, without worrying about how to actually generate the links. This achieves a better separation between specification and execution. Furthermore, we require all the declarative constraints to be satisfied, in a precise first-order logic sense, by any solution that implements the specification. Ultimately, we believe that such framework forms a better foundation for entity resolution that is transparent and high-quality while at the same time high-level.

3.2 From Declarative Constraints to Execution: Challenges

There are many foundational and architectural challenges that need to be solved, in order to achieve a functional framework for declarative entity resolution. The main research questions here will be to define precisely the language that captures all of the above types of constraints, to formulate its semantics, and to investigate the expressive power and computational aspects of the language. We outline some of the issues here, and leave further details, solutions or algorithms for future work.

The authors are grateful to Balder ten Cate, Laura Chiticariu, Mauricio A. Hernández, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa for their collaboration on various aspects of this work. This work is supported by NSF Grant IIS-0905276 and a Google Faculty Award. Part of this work was done while Tan was at IBM Research - Almaden.

References

1. Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C.: Designing and Refining Schema Mappings via Data Examples. In: SIGMOD Conference (2011)
2. Alexe, B., Chiticariu, L., Miller, R.J., Pepper, D., Tan, W.C.: Muse: a System for Understanding and Designing Mappings. In: SIGMOD Conference, pp. 1281–1284 (2008)
3. Alexe, B., Chiticariu, L., Miller, R.J., Tan, W.C.: Muse: Mapping Understanding and deSign by Example. In: ICDE, pp. 10–19 (2008)
4. Alexe, B., et al.: Simplifying Information Integration: Object-Based Flow-of-Mappings Framework for Integration. In: Castellanos, M., Dayal, U., Sellis, T. (eds.) BIRTE 2008. LNBIP, vol. 27, pp. 108–121. Springer, Heidelberg (2009)
5. Alexe, B., Hernández, M.A., Popa, L., Tan, W.C.: MapMerge: Correlating Independent Schema Mappings. PVLDB 3(1), 81–92 (2010)
6. Alexe, B., Hernández, M.A., Popa, L., Tan, W.C.: MapMerge: Correlating Independent Schema Mappings. VLDB Journal 21(1), 1–21 (2012)
7. Alexe, B., Kolaitis, P.G., Tan, W.C.: Characterizing Schema Mappings via Data Examples. In: ACM PODS, pp. 261–272 (2010)
8. Alexe, B.: Interactive and Modular Design of Schema Mappings. Ph.D. thesis, University of California, Santa Cruz (2011)
9. Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C.: Characterizing schema mappings via data examples. ACM TODS 36(4) (2011)
10. Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C.: Eirene: Interactive design and refinement of schema mappings via data examples. PVLDB (Demonstration Track) (2011)
11. Beeri, C., Vardi, M.Y.: A Proof Procedure for Data Dependencies. JACM 31(4), 718–741 (1984)
12. Bernstein, P.A., Haas, L.M.: Information Integration in the Enterprise. Commun. ACM 51(9), 72–79 (2008)
13. Microsoft BizTalk Server, http://www.microsoft.com/biztalk
14. Bonifati, A., Chang, E.Q., Ho, T., Lakshmanan, L.V.S.: HepToX: Heterogeneous Peer to Peer XML Databases (2005), http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0506002
15. Bonifati, A., Chang, E.Q., Ho, T., Lakshmanan, V.S., Pottinger, R.: HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In: VLDB, pp. 1267–1270 (2005)
16. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)
17. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. TCS 336(1), 89–124 (2005)
18. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing Schema Mappings: Second-Order Dependencies to the Rescue. TODS 30(4), 994–1055 (2005)
19. Fuxman, A., Hernández, M.A., Ho, H., Miller, R.J., Papotti, P., Popa, L.: Nested Mappings: Schema Mapping Reloaded. In: VLDB, pp. 67–78 (2006)
20. International Nucleotide Sequence Database Collection, http://www.insdc.org

21. Kolaitis, P.G.: Schema Mappings, Data Exchange, and Metadata Management. In: PODS, pp. 61–75 (2005)
22. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS, pp. 233–246 (2002)
23. Madhavan, J., Halevy, A.Y.: Composing Mappings Among Data Sources. In: VLDB, pp. 572–583 (2003)
24. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of Data Dependencies. TODS 4(4), 455–469 (1979)
25. Altova MapForce, http://www.altova.com
26. Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D.: ++spicy: an opensource tool for second-generation schema mapping and data exchange. PVLDB 4(12), 1438–1441 (2011)
27. Nash, A., Bernstein, P.A., Melnik, S.: Composition of Mappings Given by Embedded Dependencies. In: PODS, pp. 172–183 (2005)
28. Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R.: Translating Web Data. In: VLDB, pp. 598–609 (2002)
29. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4), 334–350 (2001)
30. Roth, M., Hernández, M.A., Coulthard, P., Yan, L., Popa, L., Ho, H.C.T., Salter, C.C.: XML Mapping Technology: Making Connections in an XML-centric World. IBM Sys. Journal 45(2), 389–410 (2006)
31. Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P., Lum, V.Y.: EXPRESS: A Data EXtraction, Processing, and REStructuring System. ACM Trans. Database Syst. 2(2), 134–174 (1977)
32. Smith, J.M., Bernstein, P.A., Dayal, U., Goodman, N., Landers, T.A., Lin, K.W.T., Wong, E.: Multibase: Integrating Heterogeneous Distributed Database Systems. In: AFIPS National Computer Conference, pp. 487–499 (1981)
33. Stylus Studio, http://www.stylusstudio.com
34. U.S. Census Bureau, http://www.census.gov
35. Yan, L., Miller, R., Haas, L., Fagin, R.: Data-Driven Understanding and Refinement of Schema Mappings. In: SIGMOD, pp. 485–496 (2001)
36. Yu, C., Popa, L.: Semantic Adaptation of Schema Mappings when Schemas Evolve. In: VLDB, pp. 1006–1017 (2005)

User Trust and Judgments in a Curated Database with Explicit Provenance

David W. Archer[1], Lois M.L. Delcambre[2], and David Maier[2]

[1] Galois Inc., Portland, OR 97204
[2] Portland State University, Portland, OR 97207-0751
dwa@galois.com, {lmd, maier}@cs.pdx.edu

Abstract. We focus on human-in-the-loop, information-integration settings where users gather and evaluate data from a broad variety of sources and where the levels of trust in sources and users change dynamically. In such settings, users must use their judgment as they collect and modify data. As an example, a battlefield information officer preparing a report to inform his or her superiors about the current state of affairs must gather and integrate data from many (including non-computerized) sources. By tracking multiple sources for individual values, the officer may eliminate a value from the current state whenever all of the sources where this value was found are no longer trusted. We define a conceptual model for a curated database with provenance for such settings, the Multi-granularity, Multi-provenance Model (*MMP*), which supports multiple insertions and multiple (copy-and-)paste operations for a single database element, captures the external source for all operations, and includes a Data Confidence Language that allows users to confirm or doubt values to record their atomic judgments about the data. In this paper, we briefly summarize the MMP model and show how it can be extended to support potentially complex operations including compound judgment operators (such as merging tuples to achieve entity resolution), while capturing a complete record of data provenance.

1 Introduction: Our Data-Curation Setting

Our work is motivated by our interest in a data curation setting – typically a human-in-the-loop setting – where a user is continually making judgments about the trustworthiness of data items. Green *et al.* point out that users often consider where data came from and how or by whom it has been modified in making such judgments [Green07]. As observed by Buneman *et al.*, [Buneman06] data curators are quite naturally performing information integration as they "use a wide variety of sources to select, organize, classify and annotate existing data into a database on some topic." Buneman and his colleagues also identified copy-and-paste as one of the key operations performed by data curators and noted that keeping track of the provenance due to user actions (in the form of data manipulations) is as important as keeping track of the resulting data. Their work was motivated, in part, by settings where the collective scientific community works together to evolve local copies of a single, shared database.

V. Tannen et al. (Eds.): Buneman Festschrift, LNCS 8000, pp. 89–111, 2013.

14. Cardelli, L.: The functional abstract machine. Technical Report TR-107, AT&T Bell Laboratories, Murray Hill, New Jersey (May 1983)[46]
15. Banâtre, J.P., Le Métayer, D.: A new computational model and its discipline of programming. INRIA Technical Report 566, Institut National de Recherche en Informatique et Automatique, Le Chesnay, France (1986)
16. Berry, G., Boudol, G.: The chemical abstract machine. In: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1990, pp. 81–94. ACM Press, New York (1990)[47]
17. Aït-Kaci, H.: Warren's Abstract Machine—A Tutorial Reconstruction. Logic Programming. MIT Press, Cambridge (1991)
18. Grust, T.: Monad comprehensions—a versatile representation for queries. In: Gray, P., Kerschberg, L., King, P., Poulovassilis, A. (eds.) The Functional Approach to Data Management: Modeling, Analyzing and Integrating Heterogeneous Data. Springer (September 2003)[48]
19. Bothner, P.: XQuery tutorial. Online tutorial[49]
20. Nic, M., Jirat, J.: XPath tutorial. Online tutorial[50]
21. Gesbert, N., Genevès, P., Layaïda, N.: Parametric polymorphism and semantic subtyping: the logical connection. In: Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP 2011), Tokyo Japan, September 19-21, pp. 107–116. Association for Computing Machinery, New York (2011)[51]
22. Gesbert, N., Genevès, P., Layaïda, N.: Parametric polymorphism and semantic subtyping: the logical connection. SIGPLAN Notices 46(9) (September 2011); N.B.: full version of [21]
23. Bierman, G.M., Gordon, A.D., Hri␣cu, C., Langworthy, D.: Semantic subtyping with an SMT solver. In: Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programmingm (ICFP 2010), Baltimore, MA USA, September 27-29, pp. 105–116. Association for Computing Machinery, New York (2010)[52]
24. Bierman, G.M., Gordon, A.D., Hri␣cu, C., Langworthy, D.: Semantic subtyping with an SMT solver. Journal of Functional Programming, 1–75 (2012); N.B.: full version of [23][53]
25. Jaffar, J., Maher, M.J.: Constraint Logic Programming: A survey. Journal of Logic Programming 19/20, 503–581 (1994)[54]
26. Leroy, X.: Unboxed objects and polymorphic typing. In: Proceedings of the 19th Symposium on Principles of Programming Languages (POPL 1992), pp. 177–188. Association for Computing Machinary. ACM Press (1992)[55]
27. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers—Principles, Techniques, and Tools. Addison-Wesley (1986)
28. Choe, K.M.: Personal communication. Korean Advanced Institute of Science and Technology, Seoul, South Korea (December 2000), cℎoecompiler.kaist.ac.kr

[46] http://lucacardelli.name/Papers/FAM.pdf
[47] citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.3782
[48] http://www-db.in.tum.de/ grust/files/monad-comprehensions.pdf
[49] http://www.gnu.org/software/qexo/XQuery-Intro.html
[50] http://www.zvon.org/xxl/XPathTutorial/General/examples.html
[51] http://hal.inria.fr/inria-00585686/fr/
[52] http://research.microsoft.com/apps/pubs/?id=135577
[53] http://www-infsec.cs.uni-saarland.de/~hritcu/publications/dminor-jfp2012.pdf
[54] http://citeseer.ist.psu.edu/jaffar94constraint.html
[55] http://gallium.inria.fr/ xleroy/bibrefs/Leroy-unboxed.html