Foreword

Jan Paredaens is turning 60 on the first of October, 2007. On the occasion of this birthday, we held a symposium at the University of Antwerp, on Friday 21 September 2007.

The present book contains some of the scientific papers that were presented at the symposium, but contains also additional papers, written by scientific friends of Jan, written especially to congratulate Jan, and to honor him for his contributions to the database theory research community. Also included is an interview with Jan.

Jan Van den Bussche

Contents

1 Symposium Program

2 *An interview with Jan Paredaens*
Jan Van den Bussche, Dirk Van Gucht

8 *Orderings on Annotated Collections*
Christopher Re, Dan Suciu, Val Tannen

25 *The role of graphs in computer science*
Jos Baeten, Kees van Hee

41 *A complete axiomatization for Core XPath 1.0*
Balder ten Cate, Tadeusz Litak, Maarten Marx

57 *Teaching SQL at USQ*
Stijn Dekeyser

61 *Completeness of database query languages*
George H.L. Fletcher

62 *Tree-structured object creation in database transformations*
Jan Van den Bussche

69 *Towards an axiomatization of the relational lattice*
Jan Hidders

83 *Jan Paredaens, profiel van "de prof"*
Paul De Bra

88 *Due o tre cose che mi ricordo di un viaggio a Vinci*
Bart Kuijpers

92 *Dear Jan, ...*
Geert-Jan Houben

94 *Why mathematicians make good computer scientists: reflections from a biased viewpoint*
Letizia Tanca

100 *Dear Jan, ...*
Gottfried Vossen

103 *Jan Paredaens—60*
Leonid Libkin

107 *Looking ahead and behind*
Joachim Biskup

110 *"Beste Jan," ...*
Reind van de Riet

114 *I was lucky to have Jan around*
Grzegorz Rozenberg

116 *Happy birthday, Jan!*
Victor Vianu

117 *Happy birthday, dear Jan!*
Georg Gottlob

118 Maurice Bruynooghe

119 *A database full of memories*
Andy Zaidman

120 *Pearls of modelling: from relational databases to XML suites*
Bernhard Thalheim

Symposium Program

Dirk Van Gucht (Indiana University): *The influence of the nested relational data model*

Dan Suciu (University of Washington): *Orderings on annotated complex objects*

Bernhard Thalheim (University of Kiel): *Pearls of modelling: from relational databases to XML suites*

Kees van Hee (Technical University Eindhoven): *The role of graphs in computer science*

Maarten Marx (University of Amsterdam), Jan Hidders (University of Antwerp): *Axioms, rules and operators*

George Fletcher (Washington State University): *Completeness of database query languages*

Stijn Dekeyser (University of Southern Queensland): *De gustibus et coloribus*

Bart Goethals (University of Antwerp): *Mining Jan Paredaens's social network*

Serge Abiteboul (INRIA): *Amitiés de Paris (video presentation)*

Jan Paredaens (University of Antwerp): *About research, genealogy, gastronomy and life*

An interview with Jan Paredaens

Jan Van den Bussche Dirk Van Gucht

Jan Paredaens, one of the most influential members of the database theory community, has reached the age of sixty. On this occasion, we have interviewed him about his research career, the services he has done to the research community, what drives him, and his view of the future. The interview happened in July 2007 and took place at Indiana University, Bloomington, a place which Jan has visited on numerous occasions to work with Dirk Van Gucht, getting inspiration, finding topics to work on.

In your research career, what have been the most interesting moments for you?

It all started when I was in my first year as a master's student in mathematics at the Free University of Brussels (VUB). It was around Eastern and I was looking for a topic for my master's thesis. I was mainly thinking about traditional math topics, until I ran into professor Bingen, who suggested me to check out the new topic of "computer science". We are talking here about the year 1968-69, and I hardly knew what computer science was, except of course that it had to do something with computers! Bingen sent me to talk to professor Louchard at the French-speaking University of Brussels (ULB). Louchard was a numerical analist by education, but very much interested in the then-current developments in computing. When I told Louchard that I was curious about computer science, but that I was mainly interested in formal mathematics, he connected me to his assistant, Claude Cherton. Claude gave me the lecture notes, by someone named Seymour Ginsburg, of a NATO summer school on formal languages he had attended. I still have my copy of those lecture notes, and they were my first introduction to theoretical computer science. I was very much interested, and did my master's thesis on different but equivalent variations of Turing machines, based on a JACM paper by Pat Fischer. I also did my PhD on a theoretical computer science topic, namely, stochastic automata. By adjusting the probability mechanism of a stochastic automaton, you can recognise different classes of languages in the Chomsky hierarchy and even beyond.

At the end of my PhD I was very much interested in a research career in computer science, and got a job as research scientist at the MBLE-Philips lab in Brussels (Michel Sintzoff, who already worked there at the time, was a jury member of my PhD defense). My first assignment was to work on PHOLAS, a hierarchical database system developed by Philips. I had to work on such things as fine-tuning of links and other optimisations. To be honest I did not

find that work extremely fascinating, but fortunately at EMBL we had enough freedom to pursue our own research interests on the side. In the library I stumbled upon an article, I don't remember the author, I think it appeared in IPL, that dealt with the problem of how many different keys can exist in a relation, given a set of functional dependencies. This was my first encounter with the relational database model. We were in the early 1970s, and that article must have been one of the first works that followed up on Codd's seminal papers. Given my experience with the PHOLAS system I mentioned earlier, which was complicated and messy, I was struck by the mathematical beauty and elegance of the relational database model. That encounter marked the beginning of my database theory research career. I started to work on a unifying model for various kinds of integrity constraints [10]. In the same period I also wrote my paper on the expressive power of the relational algebra [9], which later would become known as establishing the BP-completeness of the relational algebra.

After five years at MBLE, I started in 1979 as a professor at the University of Antwerp. My first two PhD students were Paul De Bra and Marc Gyssens, and I worked with them on two different kinds of decomposition of relations: Paul studied horizontal decompositions based on violations of given functional dependencies, and Marc studied vertical decompositions based on acyclic join dependencies.

I then moved on to the nested relational model, under the influence of Dirk Van Gucht. My most important result there, together with Dirk, was the "flat–flat theorem", stating that any expression in the nested relational algebra that expresses a query from flat relations to flat relations, can also be expressed in the flat relational algebra [13]. We are now speaking about the 1980s, and computer science had not stood still in the meantime. An important evolution was object orientation, under the impulse of Smalltalk, historically the most important object-oriented programming language. Also the influence of Algol 68 was very important. The confluence of all those ideas lead to the present-day prominent languages such as Java and C++. Although I never was a real programming language researcher, I was interested in the semantics of programming languages, on which topic I wrote a book together with Johan Lewi [8]. I also wrote two pedagogical books on Pascal and on C, and of course there is my book, with Paul De Bra, Marc Gyssens, and Dirk Van Gucht, on database theory [11].

So, we thought the glorious period of the relational model was over; at that time I was convinced that in 2007, relational databases would be a thing of the past and all database systems would be object-oriented. So at the end of the 1980s, beginning of the 1990s, the database research community spent a lot of energy in research on object-oriented databases, with large projects such as O_2, and companies were started up who thought they could do big business by transforming the database systems industry into object-oriented databases. History has judged otherwise, and we now see that current database systems, such as Oracle, are fundamentally still relational, although a lot of features have been added to store objects of all types, also XML documents, in relations.

My own research on object-oriented databases dealt mainly with GOOD: a

graph-oriented object database model [2], which I started with Marc Gyssens and Dirk Van Gucht. Later, no less than four PhD students joined the GOOD project: Jan Van den Bussche, Marc Andries, Marc Gemis, and Peter Peelman. With GOOD, we succeeded in modeling a number of important OO features such as generalisation and object identity. GOOD also had an entirely graphical syntax and semantics, and in retrospect I think these graphical aspects perhaps hindered, rather than helped, the communication of the model to other researchers. Looking at the present semi-structured data models, which are tree-based rather than graph-based, perhaps GOOD came before its proper time.

The next step in my research career was spatial databases, together with my then-student Bart Kuijpers. In the 1990s, people started to understand that not only general-purpose, but also special-purpose database systems were needed, such as systems that deal with text data, or sequence data, or geographical data. We studied mainly theoretical aspects of spatial databases, such as data structures [5] or topological queries [6]. This work on spatial databases also lead to constraint databases [12], because spatial data and queries can be expressed in terms of constraints over the reals. Together with Gabi Kuper and Leonid Libkin I edited a book on constraint databases [7].

After that I have been in a data mining period, mostly working on association rules, with my then-student Toon Calders [1]. Association rules are a really important concept in data mining. At around the same time, another very important concept came up in the database world, namely XML, not just as a format, but as a data structure that you want to store, query, and transform in the database. A whole bunch of languages were proposed, which in the end all culminated in the present languages XPath and XQuery. We are currently still working on these languages, with my post-doc Jan Hidders and current students Philippe Michiels and Roel Vercammen. We study expressiveness issues, and with LiXQuery [4] we made an attempt to strip XQuery of all the details that are unimportant from a theoretical perspective.

To conclude this long answer I want to say that I am indebted to all my students, who have contributed a lot to my research, and of course I have also collaborated with various people from the international database community.

You have not mentioned your work on the grammatical data model [3].

Together with Marc Gyssens and Dirk Van Gucht, I developed that model as a model for text databases, where the structure of the text plays a central role. This structure is a tree of course, and later, in our work on GOOD, we relaxed this structure to general graphs. But yes, it was a nice contribution, and if you look at that paper carefully, you already see some fundamentals of the XML data model and query languages there, "avant la lettre".

In the 1980s, you were involved in the European Association for Theoretical Computer Science (EATCS).

Yes, I became involved in EATCS through my predecessor at the University of Antwerp, Grzegorz Rozenberg, then president of the EATCS. At that time,

EATCS was legally an association according to Belgian law, and the law required a Belgian treasurer. When I succeeded him when he moved from Antwerp to Leiden, Grzegorz asked me to serve as treasurer. I remained involved in EATCS for a number of years, and served as chair of the 1984 edition of ICALP. Unfortunately, database theory never became an important topic in the EATCS community.

That might be one of the reasons why ICDT (the International Conference on Database Theory) was created?

The idea for ICDT originates at one of the workshops on Logic and Databases, organised in Toulouse by Gallaire and Minker in the mid 1980s. Together with Giorgio Ausiello and Serge Abiteboul, we started a biennial series of database theory conferences, held in historical European locations. The first was in 1986 in Rome (organised by Ausiello), the second in 1988 in Bruges (organised by myself), and the third in 1990 in Paris (organised by Abiteboul). I have served as chair of the ICDT Council for quite some time; meanwhile, that job has been taken over by Jan Van den Bussche.

Is there any advice you have to the current European database theory community? Anything you would have liked to see happen there?

The European Union funds large research projects. I would like to see more large projects focused specifically on database theory, created by collaboration among the different database theory groups in Europe, and funded by the EU.

In your office, on the wall, one can read the following quote: "Details are the jungle where the Devil hides."

Yes, this is a quote from Niklaus Wirth. He meant this in the context of programming. Every programmer knows that when programming, but even more when designing the program, there are so many details to take care of. A typical example are borderline cases: does my program work correctly for all positive integers, including the special cases 0 and 1? And so on. It is very dangerous to reason like, "well there is this one special case where my program does not work, but that case will never occur." But also when writing down formal definitions or proofs, details are extremely important. It is often easy to understand the overall argument of a formal proof, but it is often very difficult to work out all details of that proof correctly.

Are you inspired by other quotes as well?

One quote that I point out to all my student is Einstein's "A good theory should be as simple as possible, but no simpler." Well-meaning researchers often have the tendency to make their definitions and proofs more and more complicated while they understand more about the problem they are working on. (Here I am not even talking about what the lesser gods sometimes do, making their formulas too complicated just to make their research look difficult!)

It is much better, once you know you have a complete proof, to start afresh and write it down in the simplest possible form. This is extremely difficult to do well, and that is what Einstein meant. "But no simpler!" Because then you risk being simply wrong...

I have two other quotes related to mathematics. Although we use rather elementary mathematical concepts, certainly compared to what present-day pure mathematicians are working with, it must be recognised that computer science is very mathematical. Not only theoretical computer science is mathematical, also software construction is a mathematical activity. Now Grzegorz Rozenberg has said: "The mathematics of Life is computer science." Traditional sciences such as physics have been mathematical from the outset, but only in recent years also life sciences such as biology, or even psychology, are becoming more and more mathematical, and computer science has an enormous influence on these developments.

The last quote is probably the most fundamental; Richard Feynman once wrote "Why Nature is mathematical is a mystery." Yet I wonder whether perhaps there is another formalism that will eventually replace mathematics as we know it. To me, current mathematics seems insufficient to describe something as easy (to humans) as driving a car in everyday traffic. Will we ever be able to understand this process formally? Still, it is indeed wonderful how our mathematics, which we have developed since more than 2000 years, is so successful in describing Nature.

How do you think database theory will evolve in this Google era, specifically in the context of internet search?

There are two different questions here. As for the classical database theory problems, I think we may be near the end of it. Of course there will be new data models, certainly we will need to develop data models for higher dimensional data such as pictures, videos, and so on, but I think we understand well enough in general how querying a database works. I am referring here to the standard "employee-department-salary" kind of queries. This will not be much different for the newer data models. I see many more future problems left to solve in data mining: which patterns exist in the data? Which are the exceptional or otherwise "interesting" pieces of information in a large database? Security (in the sense of anti-terrorism, for example), is an increasingly important application of data mining.

Now concerning the Internet, of course that will become the largest database in the world, perhaps even a "complete" database of everything we know. In such an enormous and heterogeneous database, even supporting the classical "employee-department-salary" queries remains a difficult challenge. If you will, turning Google into a real database, yes, there we still have a lot of work to do, just like in data mining.

References

[1] T. Calders and J. Paredaens. Axiomatisation of frequent sets. *Theoretical Computer Science*, 290(1):669–693, 2003.

[2] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), 1994.

[3] M. Gyssens, J. Paredaens, and D. Van Gucht. A grammar-based approach towards unifying hierarchical data models. *SIAM Journal on Computing*, 23(6):1093–1137, 1994.

[4] J. Hidders, P. Michiels, J. Paredaens, and R. Vercammen. LiXQuery: a formal foundation for XQuery research. *SIGMOD Record*, 34(4):21–26, 2005.

[5] B. Kuijpers, J. Paredaens, and J. Van den Bussche. Lossless representation of topological spatial data. In M.J. Egenhofer and J.R. Herring, editors, *Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.

[6] B. Kuijpers, J. Paredaens, and J. Van den Bussche. On topological elementary equivalence of closed semi-algebraic sets in the plane. *Journal of Symbolic Logic*, 65(4):1530–1555, 2000.

[7] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[8] J. Lewi and J. Paredaens. *Data Structures of Pascal, Algol 68, PL/I and Ada*. Springer-Verlag, 1968.

[9] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, 1978.

[10] J. Paredaens. The interaction of integrity constraints in an information system. *Journal of Computer and System Sciences*, 52(2):357–373, 1980.

[11] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*, volume 17 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989.

[12] J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order queries on finite structures over the reals. *SIAM Journal on Computing*, 27(6):1747–1763, 1998.

[13] J. Paredaens and D. Van Gucht. Converting nested algebra expressions into flat algebra expressions. *ACM Transactions on Database Systems*, 17(1):65–93, 1992.

Orderings on Annotated Collections

Christopher Re Dan Suciu Val Tannen

September 7, 2007

We dedicate this contribution to Jan Paredaens on the occasion of his 60th birthday. For many years Jan has been a wonderful scientific mentor and collaborator, an exceptional host and a valuable friend to both of us. Jan, many thanks and we wish you many happy returns!

Dan Suciu and Val Tannen

We study order relations on collections annotated with elements from a commutative semiring. The basic question we address is: given two collections X and Y, whose elements have been annotated with values from a semiring, when is Y "more informative" than X, written $X \leq Y$? Such (pre-)orders have been studied separately for sets and for bags, but here we define them for collections annotated from an arbitrary semiring. We define a lower-, upper-, and convex- preorder, similar in spirit to the definitions on sets. The orders on sets and bags become particular instances of our general notions, by suitable choosing the semiring to be the booleans $\mathbb{B}$ (for sets) and the natural numbers $\mathbb{N}$ (for bags). Thus our definition unifies the previously separate definitions for sets and bags.

We explore deeper the connection between these order relations and properties of the semiring. To give some focus to our investigation, we address two concrete, non-trivial questions: when is the convex preorder equivalent to the conjunction of the lower and upper preorder? And when are these preorders antisymmetric? To answer the first question we introduce a new notion of *homomorphism* between collections annotated from a semiring and prove that under certain conditions the existence of a (lower-, or upper-, or convex-) homomorphism between two collections X and Y is equivalent to the (lower-, or upper-, or convex-) pre-order between X and Y. To answer the second question we introduce another characterization of the existence of homomorphisms, in terms of comparing the total annotations on filters, or on ideals respectively. In order to prove that these new definitions are equivalent to the existence of homomorphisms we show an interesting connection between homomorphisms and flows in networks, then use the min-cut max-flow theorem.

None of our results are definitive: we answer the two questions only partially, by placing certain restrictions on the semirings. We hope that future work will fully characterize the semirings for which these results continue to hold.

1 Background

1.1 Powerdomains

Denote $(A, \leq)$ an ordered set. Denote $\{A\}$ the set of finite subsets of A and $\{\!\{A\}\!\}$ the set of finite bags over A. The following pre-order relations are defined on $\{A\}$:

Lower powerdomain (Hoare) $X \leq^\flat Y$ iff $\forall a \in X.\exists b \in Y.a \leq b$.

Upper powerdomain (Smyth) $X \leq^\sharp Y$ iff $\forall b \in Y.\exists a \in X.a \leq b$.

Convex powerdomain (Plotkin) $X \leq^\natural Y$ iff both $X \leq^\flat Y$ and $X \leq^\sharp Y$.

All are pre-orders, except when $(A, \leq)$ is totally unordered. For example, if $a < b$ then $\{a, b\} \leq^\flat \{b\} \leq^\flat \{a, b\}$ and $\{a, b\} \leq^\sharp \{a\} \leq^\sharp \{a, b\}$.

1.2 The CWA and OWA Orderings

Libkin and Wong [8] define the following *closed world* and *open world* pre-order relations on $\{A\}$ and $\{\!\{A\}\!\}$:

Set ordering For $X, Y \in \{A\}$, $X \leq_{set}^{CWA} Y$ if X can be transformed into Y by a sequence of operations of the following kind:

- Replace an element a with a non-empty set B s.t. $\forall b \in B$, $a \leq b$.

Furthermore, $X \leq_{set}^{OWA} Y$ is X can be transformed into Y by a sequence of operations of the following kind:

- Replace an element a with a non-empty set B s.t. $\forall b \in B$, $a \leq b$.
- Insert some new elements.

Bag ordering For $X, Y \in \{\!\{A\}\!\}$, $X \leq_{bag}^{CWA} Y$ if X can be transformed into Y by a sequence of operations of the following kind:

- Replace an element a with an element b s.t. $a \leq b$

Furthermore, $X \leq_{bag}^{OWA} Y$ is X can be transformed into Y by a sequence of operations of the following kind:

- Replace an element a with an element b s.t. $a \leq b$
- Insert some new elements.

They proved the following two propositions:

Proposition 1.1 *Forall $X, Y \in \{A\}$:*

- $X \leq_{set}^{OWA} Y$ *iff* $X \leq^\flat Y$
- $X \leq_{set}^{CWA} Y$ *iff* $X \leq^\natural Y$.

Proposition 1.2 *If $\leq$ is an order relation on A (i.e. $\leq$ is antisymmetric) then both $\leq_{bag}^{OWA}$ and $\leq_{bag}^{CWA}$ are order relations.*

1.3 K-Collections

Green, Karvounarakis, and Tannen [6] define an algebra on relations whose tuples are annotated with elements from a commutative semiring. Foster, Green and Tannen [4] generalized this to complex objects using K-collections. In their work K was a commutative semiring $K = (K, +, \cdot, 0, 1)$: both operations $+$ and $\cdot$ were relevant for query semantics. For most of our discussion here we only need the operation $+$, and therefore it is more general to consider K to be a commutative monoid $(K, +, 0)$. Thus, K will denote a commutative monoid unless we specify explicitly that it is a semiring.

Some examples of semirings are $(\mathbb{B}, \vee, \wedge, \mathit{false}, \mathit{true})$, $(\mathbb{N}, +, \cdot, 0, 1)$, $(\mathbb{R}^+, +, \cdot, 0, 1)$, $(\mathcal{P}(E), \cup, \cap, \emptyset, E)$, and their corresponding additive monoids are $(\mathbb{B}, \vee, \mathit{false})$, etc. We abbreviate these semirings (monoids) with $\mathbb{B}$, $\mathbb{N}$, $\mathbb{R}^+$, $\mathcal{P}(E)$.

Definition 1.3 *Let A be a set. The* support *of a function $X : A \to K$ is the set $\{a \mid X(a) \neq 0\}$. A* K-collection *over A is a function $X : A \to K$ with finite support. Denote by $Coll_K(A)$ the set of K-collections over A.*

Define 0 to be the collection $0(k) = 0, \forall k \in K$, for any two collections X, Y, define the collection $X + Y$ by $(X + Y)(k) = X(k) + Y(k)$, $\forall k \in K$.

$\mathbb{B}$-collections over A are finite sets of elements from A, while the $\mathbb{N}$-collections to the finite bags: thus, $Coll_{\mathbb{B}}(A) = \{A\}$ and $Coll_{\mathbb{N}}(A) = \{\!\{A\}\!\}$.

For another example of interest, fix a set V of boolean variables, and denote $BoolExp(V)$ the set of boolean expressions over V up to logical equivalence. Then $(BoolExp(V), \vee, \wedge, \mathit{false}, \mathit{true})$ is a semiring. $BoolExp(V)$-collections are a particular case of the c-tables of [7], used under the name *boolean c-tables* in [5]. Yet another example is provided by the events of a probability space $(\mathcal{P}(E), \cup, \cap, \emptyset, E)$; we assume here that $\mathcal{P}(E)$ is a finite sample space and that *all* its subsets are measurable events. The last two examples are, in fact, boolean algebras. Any boolean algebra, and more generally, any bounded distributive lattice is also a commutative semiring.

Definition 1.4 *Let $X \in Coll_K(A)$. Its* weight *is $w(X) = \sum_{a \in A} X(a)$. (Since X has finite support, this sum is well-defined even when A is infinite.)*

To illustrate, the weight of a bag is the number of its elements, the weight of an $\mathbb{R}^+$-collection is the sum of the annotations of all elements, and the weight of an ordinary finite set is a boolean value indicating whether the set is non-empty.

Example 1.5 A finite probability space is a pair (Ω, μ) where Ω is a finite set and $\mu : \Omega \to [0, 1]$ such that $\sum_{\omega \in \Omega} \mu(\omega) = 1$. A finite probability space can be viewed as an annotated collection with weight 1: $X \in Coll_{\mathbb{R}^+}(\Omega)$ s.t. $w(X) = 1$.

We denote a K-annotated collection X as a set $X = \{(a_1, k_1), \ldots, (a_n, k_n)\}$, where $\{a_1, \ldots, a_n\}$ is the support of X, and $k_i = X(a_i)$, for $k = 1, \ldots, n$.

2 Ordered Monoids

Let $(K, +, 0)$ be a monoid. Define $k_1 \leq k_2$ if $\exists k.k_1 + k = k_2$. One can check that $\leq)$ is a pre-order, that $0 \leq k$ for any $k \in K$; when K is a semiring then $\cdot$ is monotone in each argument. Call the monoid *naturally ordered* if $(K, \leq)$ is an order, i.e. if $k_1 \leq k_2$ and $k_2 \leq k_1$ implies $k_1 = k_2$ (antisymmetry).

$\mathbb{B}, \mathbb{N}, \mathbb{R}^+$, $BoolExp(V)$, and $\mathcal{P}(E)$ are all naturally ordered. For $\mathbb{B}$, and $BoolExp(V)$ the order corresponds to logical implication. For $\mathcal{P}(E)$ it's simply set inclusion. For $\mathbb{N}$ and $\mathbb{R}^+$ the order corresponds to the usual numerical order. $\mathbb{Z}$ is not naturally ordered, because $x \leq y$ forall $x, y \in \mathbb{Z}$.

Recall that when the monoid is idempotent (i.e. $x + x = x$) then $\leq$ is an order relation, and that $(K, \leq)$ forms a semi-lattice with $k_1 \vee k_2 = k_1 + k_2$: thus, every idempotent monoid is naturally ordered.

We will assume throughout the paper that the monoid K is naturally ordered, unless otherwise stated.

3 Pre-Orders on K-collections

3.1 A Pointwise Order

We start by examining how we can order the set $Coll_K(A)$ assuming no prior order on the domain A. "When is a collection Y more informative than X, in notation $X \leq Y$?" A natural answer is "when each element in Y has a more informative annotation than in X", which leads us to the pointwise order:

$$X \leq Y \quad \Leftrightarrow \quad \forall a \in A.\ X(a) \leq Y(a)$$

It is easy to check that this is an order relation.

Another obvious idea for defining a pre-order on $Coll_K(A)$ is to observe that $(Coll_K(A), +, 0)$ is a monoid, for which we have defined a pre-order in the previous section. This order, however, coincides with the one defined pointwise:

Proposition 3.1 *$X \leq Y$ iff $\exists Z \in Coll_K(A).\ X + Z = Y$.*

For finite sets (which are $Coll_{\mathbb{B}}(A)$) the pointwise order is set inclusion; for finite bags (which are $Coll_{\mathbb{N}}$) it is bag inclusion. As we shall see, the situation is more interesting with the other monoids we have considered.

We end this section with a simple, alternative characterization of the pointwise order on K-collections:

Proposition 3.2 *The pointwise order on $Coll_K(A)$ is the smallest pre-order satisfying the following two properties:*

- *If $X \leq Y$ and $X' \leq Y'$ then $X + Y \leq X' + Y'$*
- $0 \leq X$.

3.2 Application: c-Tables

A c-table T is a set where each element is annotated with a boolean expression [7], thus $T \in Coll_{BoolExp(V)}(A)$. What does the point-wise order on c-tables mean ? A c-table represents an *incomplete database*, i.e. a set of finite subsets of A denoted $Mod(T)$ and formally defined as follows. Recall that T is a function $T : A \to BoolExp(V)$; then, given a truth assignment $\nu : V \longrightarrow \mathbb{B}$ and denoting $\bar{\nu} : BoolExp(V) \to \mathbb{B}$ its extension to boolean expressions, we have $\bar{\nu} \circ T : A \to \mathbb{B}$, thus $\bar{\nu} \circ T \in Coll_{\mathbb{B}}(A)$. Then:

$$Mod(T) \quad = \quad \{\bar{\nu} \circ T \mid \nu : V \longrightarrow \mathbb{B}\} \subseteq Coll_{\mathbb{B}}(A)$$

Proposition 3.3 *Let T_1, T_2 be two $BoolExp(V)$-collections. Then*

$$T_1 \leq T_2 \quad \Rightarrow \quad Mod(T_1) \leq^{\natural} Mod(T_2)$$

(convex powerdomain ordering)

Proof: Since the order on $BoolExp(V)$ corresponds to logical implication, for each truth assignment ν and each $a \in A$ we have that $\bar{\nu} \circ T_1(a) = true$ implies $\bar{\nu} \circ T_2(a) = true$ hence $\nu \circ T_1 \subseteq \nu \circ T_2$. The Plotkin (convex powerdomain) ordering follows. □

Thus, the order relation on c-tables implies that the incomplete databases they represent are ordered by the convex powerdomain order. Note that the converse doesn't hold[1]. For example, let $X, Y \in V$ be two distinct boolean variables, $a \in A$ an element in the domain, and consider the following two c-tables: $T_1 = \{(a, X)\}$ and $T_2 = \{(a, Y)\}$. Both have support $\{a\}$, but one annotates a with X the other with Y. Then $T_1 \not\leq T_2$, yet $Mod(T_1) = Mod(T_2) = \{\emptyset, \{a\}\}$.

3.3 Lower, Upper, and Convex Pre-Orders

The more interesting case that we study is when the domain is already ordered. Thus, we consider an ordered set $(A, \leq)$, and examine how to order the K-collections over A. Clearly we want the pre-order on collections to be closed under addition (like in Proposition 3.2), but now we also want to allow to "replace" an element $a \in A$ with some element $b \in A$ s.t. $a \leq b$. More precisely, call a binary relation $\preceq$ on $Coll_K(A)$ *closed* if it satisfies:

- If $a, b \in A, a \leq b$, then forall $k \in K$, $\{(a, k)\} \preceq \{(b, k)\}$
- If $X \preceq Y$ and $X' \preceq Y'$ then $X + Y \preceq X' + Y'$.

Definition 3.4 *Given an ordered set $(A, \leq)$, define the following three pre-orders on $Coll_K(A)$:*

[1] This was pointed to us by T.J. Green.

Convex pre-order $\leq_K^\natural$ *is the smallest closed pre-order.*

Lower pre-order $\leq_K^\flat$*, is the smallest closed pre-order satisfying* $0 \leq_K^\flat X$*.*

Upper pre-order $\leq_K^\sharp$*, is the smallest closed pre-order satisfying* $X \leq_K^\sharp 0$*.*

A direct consequence of this definition is the following characterization extending Proposition 1.1:

Proposition 3.5 *The following hold:*

- $X \leq_{set}^{OWA} Y$ *iff* $X \leq^\flat Y$ *(Hoare) iff* $X \leq_{\mathbb{B}}^\flat Y$*.*
- $X \leq_{set}^{CWA} Y$ *iff* $X \leq^\natural Y$ *(Plotkin) iff* $X \leq_{\mathbb{B}}^\natural Y$*.*
- $X \leq_{bag}^{OWA} Y$ *iff* $X \leq_{\mathbb{N}}^\flat Y$
- $X \leq_{bag}^{CWA} Y$ *iff* $X \leq_{\mathbb{N}}^\flat Y$ *and* $w(X) = w(Y)$*.*

Thus, our definitions extend naturally the familiar pre-orders on sets and bags. Furthermore, the following property is easy to check:

Proposition 3.6 *The three pre-order relations affect the weight as follows:*

- $X \leq_K^\natural Y$ *implies* $w(X) = w(Y)$*.*
- $X \leq_K^\flat Y$ *implies* $w(X) \leq w(Y)$*.*
- $X \leq_K^\sharp Y$ *implies* $w(X) \geq w(Y)$*.*

Thus, on sets $\leq^\natural$ preserves the empty/non-empty property, while on bags it preservers the total number of elements in the bags.

4 Two Questions

To give some focus to our investigation, we address two non-trivial questions:

- Is $X \leq_K^\natural Y$ equivalent to $X \leq_K^\flat Y$ and $X \leq_K^\sharp Y$? On sets the two were equivalent by definition. On bags $\leq^\natural$ hadn't been defined before, but one can check that with our definition the convex pre-order is indeed equivalent to the lower- and upper-order[2]. But in general, the answer to this question is non-obvious: one needs to show that if one can derive $X \leq Y$ by using the rule $0 \leq U$, and one can also derive $X \leq Y$ by using the rule $V \leq 0$, then one can derived $X \leq Y$ without using any of these two rules.

[2]We prove this below.

- When is $\leq_K^\natural$ an order, i.e. antisymmetric ? We have seen that it is always an order on bags, and that it is also an order on sets when then underlying domain is totally unordered. We would like to understand under which restrictions on K, on $(A, \leq)$, and/or on the K-collections is $\leq_K^\natural$ antisymmetric.

Ideally we would like to give necessary and sufficient conditions on the mononid K under which these two questions have positive answers. Instead, we answer both questions only partially, and leave a full characterization open. Perhaps more interesting than the actual questions are the tools and notions that we develop along the way.

5 Is Convex = Upper- AND Lower ?

Clearly $\leq_K^\natural$ implies both $\leq_K^\flat$ and $\leq_K^\sharp$. In general the converse fails, as the following example shows.

Example 5.1 We show an example where $X \leq_{K_1}^\flat Y$ and $X \leq_{K_1}^\sharp Y$ but $X \not\leq_{K_1}^\natural Y$. Define the monoid $(K_1, \cup, \emptyset)$ where K_1 is the following subset of $\mathcal{P}(E)$ where $E = \{1,2,3,4\}$: $K_1 = \mathcal{P}(E) - \{k \mid |k| = 1\}$. Thus, K_1 contains all sets except the singleton sets: it is closed under union, but not under intersection. Let $A = \{a, b, c, d\}$ ordered by: $a < c, a < d, b < c, b < d$. Consider the annotated collections $X = \{(a, 12), (b, 34)\}$ and $Y = \{(c, 14), (d, 23)\}$. Here 12 denotes $\{1, 2\}$, etc. Note that $\{(a, 12)\} \leq_{K_1}^\flat \{(a, 1234)\}$ because of $\{(a, 12)\} + 0 \leq_{K_1}^\flat \{(a, 12)\} + \{(a, 34)\}$. Using this, we perform the two dual derivations:

$$\begin{aligned} X \leq_{K_1}^\flat \{(a, 1234), (b, 1234)\} &= \{(a, 14), (b, 23)\} + \{(a, 23), (b, 14)\} \\ &\leq_{K_1}^\natural Y + Y = Y \end{aligned}$$

$$\begin{aligned} X = X + X \leq_{K_1}^\natural \{(c, 12), (d, 34)\} + \{(c, 34), (d, 12)\} \\ = \{(c, 1234), (d, 1234)\} \leq_{K_1}^\sharp Y \end{aligned}$$

This shows that $X \leq_{K_1}^\flat Y$ and $X \leq_{K_1}^\sharp Y$. On the other hand one can prove by induction on the derivation of $U \leq_{K_1}^\natural V$ that, if $U(a) = 12$ then either $V(a) = 12$ or $V(c) \subseteq 12$ or $V(d) \subseteq 12$, thus, $X \not\leq_{K_1}^\natural Y$. This is because 12 cannot be split into $1 + 2$ in K_1, thus we can only keep it as annotation on a or move it up to c or d.

Thus, the convex pre-order and the conjunction of the lower- and upper-pre-order do not coincide in general. We show, however, that they do coincide on two interesting classes of monoids: strict monoids, and distributive lattices.

Definition 5.2 *A naturally ordered monoid $(K, +, 0)$ is* strict *if forall x, y, $x + y = y$ implies $x = 0$.*

Strictness is a weaker condition than cancellation [1], which requires $x+y = x'+y$ to imply $x = x'$. Examples of strict monoids (in fact cancellative monoids) are $\mathbb{N}$ and $\mathbb{R}^+$; all lattices on the other hand are non-strict, e.g. $\mathbb{B}$, $\mathcal{P}(E)$ and $BoolExpr(V)$.

Theorem 5.3 *If either (1) $(K,+,0)$ is a strict monoid, or (2) $(K,+,\cdot,0,1)$ is a distributive lattice, then the following holds:*
$X \leq_K^\natural Y$ *iff* $X \leq_K^\flat Y$ *and* $X \leq_K^\sharp Y$.

The case when K is strict follows from the following stronger property:

Proposition 5.4 *If $(K,+,0)$ is a naturally ordered, strict monoid and $w(X) = w(Y)$ then $X \leq_K^\natural Y$ iff $X \leq_K^\flat Y$ iff $X \leq_K^\sharp Y$.*

This holds because even a single application of the rule $0 \leq_K^\flat X$ or $X \leq_K^\sharp 0$ strictly increases/decreases the weight.

In the remainder of this section we prove Theorem 5.3 for the case when K is a distributive lattice. To do that we introduce an alternative characterization of the three pre-orders, in terms of "homomorphisms", which is of interest in itself.

5.1 Homomorphisms between K-Collections

We give here an alternative characterization of the three pre-order relations on K-collections in terms of the existence of certain homomorphisms between the collections.

Fix a naturally ordered monoid $(K,+,0)$, and an ordered domain $(A,\leq)$. We identify a K-collection $X : A \to K$ with an array $X = (x_a)_{a\in A}$ with finite support. A *matrix* $H = (h_{ba})_{a,b\in A}$ is a function $H : A \times A \to K$ with finite support. The matrix is *order compatible* if $h_{ba} \neq 0 \Rightarrow a \leq b$.

Definition 5.5 *Let $X, Y \in Coll_K(A)$, and H be an order-compatible matrix.*

Convex homomorphism *H is a convex homomorphism, $H : X \to^\natural Y$ if:*

$$\forall a.(\sum_{b\in A} h_{ba} = x_a) \quad \text{and} \quad \forall b.(\sum_{a\in A} h_{ba} = y_b)$$

Lower homomorphism *H is a lower homomorphism, $H : X \to^\flat Y$ if:*

$$\forall a.(\sum_{b\in A} h_{ba} \geq x_a) \quad \text{and} \quad \forall b.(\sum_{a\in A} h_{ba} = y_b)$$

Upper homomorphism *H is an upper homomorphism, $H : X \to^\sharp Y$ if:*

$$\forall a.(\sum_{b\in A} h_{ba} = x_a) \quad \text{and} \quad \forall b.(\sum_{a\in A} h_{ba} \geq y_b)$$

We will show that the existence of a convex (lower, upper) homomorphism is equivalent to the convex (lower, upper) pre-order. One direction is simple. If there exists a convex homomorphism $H : X \to^\natural Y$ then $X \leq^\natural_K Y$. Indeed, for every pair a, b s.t. $h_{ba} \neq 0$ denote the singleton K-collections $X^{a,b} = \{(a, h_{ba})\}$ and $Y^{a,b} = \{(b, h_{ba})\}$. We have $X^{a,b} \leq^\natural_K Y^{a,b}$ because $a \leq b$. Also, $X = \sum_{a,b} X^{a,b}$, $Y = \sum_{a,b} Y^{a,b}$, hence $X \leq^\natural_K Y$. The same holds for the other two pre-orders: for example if $H : X \to^\flat Y$ then there exists Z s.t. $H : X + Z \to^\natural Y$ and thus $X \leq^\flat_K X + Z \leq^\natural_K Y$.

The other direction holds if and only if the monoid is a *refinement monoid* [1].

Definition 5.6 *K is a* refinement monoid *if forall x_1, x_2, y_1, y_2 s.t. $x_1 + x_2 = y_1 + y_2$ there exists $(h_{ij})_{1 \leq i,j \leq 2}$ (called* refinements *of x_1, x_2, y_1, y_2) s.t.*

$$\begin{array}{ll} x_1 = h_{11} + h_{21} & x_2 = h_{12} + h_{22} \\ y_1 = h_{11} + h_{12} & y_2 = h_{21} + h_{22} \end{array}$$

We now give examples of refinement monoids.

Proposition 5.7 *Let $(K, +, \cdot, 0, 1)$ be a lattice then $(K, +, 0)$ is a refinement monoid if and only if K is a distributive lattice.*

Proof: First, for any distributive lattice $(K, +, 0)$ is a refinement monoid: simply take $h_{ij} = y_i \cdot x_j$. If K is not distributive then by Birkhoff's theorem, there is a sublattice of K isomorphic to $M_3 = \{\bot, \top, u, v, w\}$ partially ordered by $\bot \leq \{u, v, w\} \leq \top$ or $N_5 = \{\bot, u, v_1, v_2, \top\}$ ordered as $\bot \leq u \leq \top$ and $\bot \leq v_1 \leq v_2 \leq \top$. For M_3 consider the sum above with $x_1 = u$, $x_2 = v$, $y_1 = w$ and $y_2 = v$, *i.e.* $u + v = w + v = \top$. Since $h_{11} \leq u \wedge w = \bot$, we have that $h_{21} = u$. This in turn implies that there must be l to satisfy $v = u + l$, but no such l can exist. For N_5 we inspect the sum $u + v_1 = u + v_2$, it is immediate that $h_{21} = h_{12} = \bot$ which implies that $v_1 = h_{22}$ but also $v_2 = h_{22}$, a contradiction. □

There are other examples of refinement monoids as well.

Proposition 5.8 *The monoids $\mathbb{N}$ and $\mathbb{R}^+$ are refinement monoids.*

Proof: The solutions h_{ij} are given by:

$$\begin{array}{ll} h_{11} = y_1 - x_2 + p & h_{12} = x_2 - p \\ h_{21} = y_2 - p & h_{22} = p \end{array}$$

for any parameter p s.t. $x_2 - y_1 \leq p \leq \min(x_2, y_2)$. □

For a trivial illustration we can refine $10 + 5 = 7 + 8$ into the matrix $\begin{pmatrix} 2 & 5 \\ 8 & 0 \end{pmatrix}$ obtained by choosing $p = 0$.

Example 5.9 We give two examples of non-refinement monoids. One is the monoid K_1 in Example 5.1: $12 + 34 = 14 + 23$ yet we cannot refine them because we miss all singleton sets. For another example, define $(\mathbb{N}^1, +, 0)$ to be the monoid over the set $\mathbb{N}^1 = \mathbb{N} - \{1\}$: we have $3 + 3 = 2 + 4$ yet we cannot refine them because we miss 1.

We now show that being a refinement monoid is both a necessary and sufficient condition for the other direction to hold. We start with necessity.

Proposition 5.10 *If the implication $X \leq_K^{\natural} Y \Rightarrow \exists H : X \to^{\natural} Y$ holds then K is a refinement monoid.*

Proof: Let $(A, \leq)$ consists of five elements a, b, c, d, e ordered as follows: $\{a, b\} < c < \{d, e\}$ (i.e. $a < c, b < c, c < d, c < e$). For any $x_1 + x_2 = y_1 + y_2 = z$ define $X = \{(a, x_1), (b, x_2)\}$, $Y = \{(d, y_1), (e, y_2)\}$. Then $X \leq_K^{\natural} \{(c, z)\} \leq_K^{\natural} \{(d, y_1), (e, y_2)\}$. Thus, there exists a homomorphism $G : X \to^{\natural} Y$. Define $h_{11} = g_{ad}$, $h_{12} = g_{ae}$, $h_{21} = g_{bd}$, $h_{22} = g_{be}$. One can check that all other entries in G are 0[3], and therefore one can show that $H = (h_{ij})_{ij}$ is a refinement, which proves that K is a refinement monoid. □

We prove now sufficiency, and for that we fix a refinement monoid K.

Lemma 5.11 *The following statement $S(m, n)$ holds forall $m, n \geq 1$:*

$$S(m,n) : (x_1 + \ldots + x_n = y_1 + \ldots + y_m) \Longrightarrow$$
$$\text{there exists } H = (h_{ij})_{i=1,m,j=1,n} \text{ s.t. } x_j = \sum_i h_{ij} \text{ and } y_i = \sum_j h_{ij}$$

Proof: $S(1, n)$ holds trivially: take $h_{1j} = x_j$; similarly $S(m, 1)$. $S(2, 2)$ holds because K is a refinement monoid. To show $S(m, n + 1)$, denote X' the vector $x'_n = x_n + x_{n+1}$, $x'_j = x_j$ for $j \neq n$. By induction $S(m, n)$ holds, so let H' be the refinement of X' and Y. Its last column satisfies $\sum_i h'_{in} = x_n + x_{n+1}$: by induction $S(m, 2)$ holds, hence we can find $h_{in}, h_{i(n+1)}$ s.t. $h'_{in} = h_{in} + h_{i(n+1)}$, $x_n = \sum_i h_{in}$, and $x_{n+1} = \sum_i h_{i(n+1)}$. Denote H the matrix is obtained from H' by replacing the last column with two columns, $(h_{in})_i$ and $(h_{i(n+1)})_i$. Obviously H is a refinement of X and Y. □

Lemma 5.12 *Let K be a refinement monoid and $H = (h_{ba})_{a,b \in A}, G = (g_{cb})_{b,c \in A}$ two order-compatible matrices s.t. $\forall b \in A, \sum_a h_{ba} = \sum_c g_{cb}$. Then there exists a (not necessarily unique) order compatible matrix $L = (l_{ca})_{a,c \in A}$ s.t. $\forall a \in A, \sum_c l_{ca} = \sum_b h_{ba}$ and $\forall c \in A, \sum_a l_{ca} = \sum_b g_{cb}$. We call L a* refinement *of H and G.*

Proof: For each $b \in A$ construct two vectors $H^b = (h_{ba})_{a \in A} = (h^b_a)_{a \in A}$ and $G^b = (g_{cb})_{c \in A} = (g^b_c)_{c \in A}$. Since $\sum_a h^b_a = \sum_c g^b_c$ we apply Lemma 5.11 to obtain

[3]For example $g_{ac} + g_{bc} = y_c = 0$, which implies $h_{ac} = h_{bc} = 0$ because K is naturally ordered.

a refinement $L^b = (l^b_{ca})_{a,c \in A}$ of H^b and G^b. L^b is order compatible, because $h^b_a \neq 0$ implies $a \leq b$ and $g^b_c \neq 0$ implies $b \leq c$, hence the only non-zero entries l^b_{ca} are for $a \leq b \leq c$. Define $L = \sum_b L^b$. We show that L is a refinement of H and G. For every $a \in A$ we have $\sum_c l_{ca} = \sum_c \sum_b l^b_{ca} = \sum_b \sum_c l^b_{ca} = \sum_b h^b_a = \sum_b h_{ba}$. Similarly, for every $c \in A$, $\sum_a l_{ca} = \sum_b g_{cb}$.

□

Theorem 5.13 *Let K be a refinement monoid. Then:*

- *$X \leq^\natural_K Y$ iff there exists $H : X \to^\natural Y$.*
- *$X \leq^\flat_K Y$ iff there exists $H : X \to^\flat Y$.*
- *$X \leq^\sharp_K Y$ iff there exists $H : X \to^\sharp Y$.*

Proof: We have already shown the "if" part. We show here "only if", for the convex pre-order only: the others are similar. We wll construct a homomorphism $H : X \to^\natural Y$ by induction on the length of the derivation $X \leq^\natural_K Y$. For the base case $\{(a,k)\} \leq^\natural_K \{(b,k)\}$ define $H = \{((a,b),k)\}$: it is order-compatible because $a \leq b$. For $X + Y \leq^\natural_K X' + Y'$ construct inductively the homomorphisms $H : X \to^\natural Y$ and $G : X' \to^\natural Y'$ and define $L = H + G$: clearly $L : X + X' \to^\natural Y + Y'$. Finally, we need to show how to construct H for the reflexivity rule $X \leq^\natural_K X$ (take H to be the identity on the support of X) and for the transitive closure rule: $X \leq^\natural_K Y$ and $Y \leq^\natural_K Z$ implies $X \leq^\natural_K Z$: here take the homomorphism to be a refinement of the homomorphisms for $X \leq^\natural_K Y$ and $Y \leq^\natural_K Z$. □

Finally, we prove case (2) of Theorem 5.3. Suppose $X \leq^\flat_K Y$ and $X \leq^\sharp_K Y$. Then we have two homomorphisms: $H : X \to^\flat Y$ and $G : X \to^\sharp Y$. Define L to be $l_{ba} = h_{ba} \cdot x_a + g_{ba} \cdot y_b$. We verify that $L : X \to^\natural Y$ by direct calculation.

$$\begin{array}{rccc} & h_{ba} \cdot x_a & \leq h_{ba} \cdot x_a + g_{ba} \cdot y_b \leq & x_a + g_{ba} \\ x_a = (\sum_b h_{ba}) \cdot x_a & & \leq \sum_b l_{ba} \leq & x_a + \sum_b g_{ba} = x_a \end{array}$$

This shows $\sum_b l_{ba} = x_a$. Similarly $\sum_a l_{ba} = y_b$.

5.2 Application: Ordered Probabilistic Spaces

Recall that a finite probability space is a pair (Ω, μ) where Ω is a finite space and μ is a probability measure on Ω. Equivalently, the set of all measures on Ω is a collection $X \in Coll_{\mathbb{R}^+}(\Omega)$ s.t. $w(X) = 1$. Suppose that we are also given an order $\leq$ on Ω that intuitively tells us when one element of Ω is "more informative" than another. The question is then: When is a probability measure X "more informative" than a probability measure Y ?

We have three pre-orders $\leq^\natural_{\mathbb{R}^+}$, $\leq^\flat_{\mathbb{R}^+}$, and $\leq^\sharp_{\mathbb{R}^+}$. Since $\mathbb{R}^+$ is strict ($x + y = y$ implies $x = 0$) by Proposition 5.4 these three orders coincide. How can we check

if $X \leq_{\mathbb{R}^+} Y$? Since $\mathbb{R}^+$ is also a refinement monoid, we can apply Theorem 5.13 and check for the existence of a homomorphism H. If we further normalize the matrix H to $\tilde{H}$, by setting $\tilde{h}_{ij} = h_{ij}/x_j$ if $x_j \neq 0$, and $\tilde{h}_{ij} = \delta_{ij}$ otherwise[4], then $\tilde{H}$ is a stohastic matrix (i.e. $\forall j, \sum_i \tilde{h}_{ij} = 1$), and, furthermore, $Y = \tilde{H}X$.

To illustrate, let $A = \{a, b\}$ and $\Omega = \{A\}$, hence there are four elements of Ω: $\emptyset, \{a\}, \{b\}, \{a, b\}$. Assuming that larger sets are more informative, the order on Ω is given by set inclusion. A measure X assigns a probability to each of the four sets, e.g. $X = (1/4, 1/4, 1/4, 1/4)$ corresponds to independently choosing the elements a and b with probability $1/2$. Then $X \leq^{\natural}_{\mathbb{R}^+} Y$ means that Y can be obtained from X by "moving probability mass up". For example, let $Y = (0, 1/3, 1/3, 1/3)$: this corresponds to independently choosing a and b each with probability $1/2$ and conditioning on the set being non-empty. Then $X \leq^{\natural}_{\mathbb{R}^+} Y$ because $Y = \tilde{H}X$, for the following stohastic matrix:

$$\tilde{H} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 0 \\ 1/3 & 0 & 0 & 1 \end{pmatrix}$$

Thus, we have an interesting pre-order relation on probabilistic sets given by the existence of an order-compatible stohastic matrix mapping one set into the other. Even more interesting, this pre-order is actually an order relation: this non-obvious fact is a consequence of our result in the next section.

6 When is $\leq^{\natural}_K$ an Order Relation ?

We have seen that for sets $\leq^{\natural}$ is an order when the underlying domain is a totally unordered set, while for bags, $\leq^{\natural}_{\mathbb{N}}$ is always an order relation (Proposition 1.2). We study when $\leq^{\natural}_K$ is an order relation.

Fix an ordered set $(A, \leq)$. An *ideal* over $(A, \leq)$ is a subset $S \subseteq A$ s.t. $\forall a \in S$, if $b \leq a$ then $b \in S$. A *filter* is an ideal in the dual order $(A, \geq)$. For a set S we denote $(S] = \{b \mid \exists a \in S. b \leq a\}$ ($[S) = \{b \mid \exists a \in S. a \leq b\}$) the ideal (filter) generated by S. We denote $[a) = [\{a\})$ ($(a] = (\{a\}]$) and call it *principal filter* (*principal ideal*). We denote:

- $F(A)$ = the filters of A.
- $f(A)$ = the principal filters of A.
- $I(A)$ = the ideals of A.
- $i(A)$ = the principal ideals of A.

Fix now a naturally ordered monoid K and let $X \in Coll_K(A)$. For any $S \subseteq A$, denote $X(S) = \sum_{a \in S} X(a)$.

[4] $\delta_{ii} = 1$, $\delta_{ij} = 0$ for $i \neq j$.

Definition 6.1 *We define four pre-order relations on $Coll_K(A)$:*

Weak F-pre-order $X \leq_K^f Y$ *if forall* $F \in f(A)$, $X(F) \leq Y(F)$.

F-pre-order $X \leq_K^F Y$ *if forall* $F \in F(A)$, $X(F) \leq Y(F)$.

Weak I-pre-order $X \leq_K^i Y$ *if forall* $I \in i(A)$, $X(I) \geq Y(I)$.

I-pre-order $X \leq_K^I Y$ *if forall* $I \in I(A)$, $X(I) \geq Y(I)$.

The F-pre-order implies the weak F-pre-order, $X \leq_K^F Y \Rightarrow X \leq_K^f Y$, and similarly for the I-pre-orders. The converse does not hold in general, as the following example shows, but holds for all idempotent monoids:

Example 6.2 Consider the monoid $\mathbb{R}^+$, let $A = \{a, b, c, d\}$ be the diamond order: $a < b < d$ and $a < c < d$, and let $X = (0, 2, 2, 0)$ (i.e. $X = \{(b, 2), (c, 2)\}$) $Y = (1, 1, 1, 1)$. Then $X \leq_K^f Y$ because the principal filters are $\{a, b, c, d\}, \{b, d\}, \{c, d\}, \{d\}$ and their image under X, Y are $(4, 2, 2, 0)$ and respectively $(4, 2, 2, 1)$. However $X \not\leq_K^F Y$ because on the (non-principal) filter $F = \{b, c, d\}$ we have $X(F) = 4$ while $Y(F) = 3$.

Proposition 6.3 *If* $(K, +, 0)$ *is idempotent then* $X \leq_K^f Y$ *implies* $X \leq_K^F Y$ *and* $X \leq_K^i Y$ *implies* $X \leq_K^I Y$.

Proof: Let $F \in F(A)$. Then $F = \bigcup_{a \in F}[a)$, which implies that $X(F) = \sum_{a \in F} X([a))$, because $+$ is idempotent. Both statements follow from here. □

6.1 Relating F- and I-pre-orders to Homomorphisms

We examine now the relationship between the filter/ideal pre-orders and the homomorphisms defined in Sec. 5.1: recall that the latter are equivalent (under certain conditions) to the lower and upper pre-orders. First we show:

Proposition 6.4 *(1) Let* $H : X \to^\flat Y$ *be a lower homomorphism; then* $X \leq_K^F Y$. *(2) Let* $H : X \to^\sharp Y$ *be an upper homomorphism; then* $X \leq_K^F Y$.

Proof: Fix a lower homomorphism H and a filter $F \subseteq A$:

$$\begin{aligned} X(F) &= \sum_{a \in F} x_a \leq \sum_{a \in F} \sum_{b \in A : b \geq a} H_{ba} \\ &= \sum_{b \in F} \sum_{a \in F, a \leq b} H_{ba} \\ &\leq \sum_{b \in F} \sum_{a \in A : a \leq b} H_{ba} = \sum_{b \in F} Y_b = Y(F) \end{aligned}$$

□

The converse is more difficult, and we have shown it only for two cases:

Theorem 6.5 *Suppose K satisfies one of the following two conditions: (1) K is a distributive lattice or (2) $K = \mathbb{R}^+$. If $X \leq^F_K Y$ then there exists $H : X \rightarrow^\flat Y$. Dually, if $X \leq^I_K Y$ then there exists $H : X \rightarrow^\sharp Y$.*

This allows us to answer our question "when is $\leq^\flat_K$, or $\leq^\sharp_K$ an order relation ?" by answering it for $\leq^F_K$ or $\leq^I_K$ instead. Given a collection $X \in Coll_K(A)$ we can construct a collection $\bar{X} \in Coll_K(F(A))$ by defining $\bar{X}(F) = X(F)$ for any filter F. Then $X \leq^F_K Y$ iff $\bar{X} \leq \bar{Y}$: the latter is a pointwise order, hence it is an order on $Coll_K(F(A))$, i.e. antisymmetric. When the mapping $X \rightsquigarrow \bar{X}$ is injective, then this implies that $\leq^F_K$ is also antisymmetric. We give next a sufficient condition for $X \rightsquigarrow \bar{X}$ to be injective.

Let $x, y \in K$ where K is a naturally ordered monoid, and $x \leq y$. If the set $\{z \mid x + z = y\}$ has a glb z_0 and $x + z_0 = y$ then we call z_0 the difference of x and y, denoted $x - y$. In particular, the difference $x - 0$ exists forall x, and $x - 0 = x$. Call $X \in Coll_K(A)$ *non-redundant* if forall $a \in A$, $X(a) = X([a)) - X([a) - \{a\})$.

Theorem 6.6 *Suppose X, Y are non-redundant. Then $X \leq^F_K Y$ and $Y \leq^F_K X$ implies $X = Y$.*

Proof: We have $\bar{X} \leq \bar{Y}$ and $\bar{Y} \leq \bar{X}$, hence $\bar{X} = \bar{Y}$, i.e. the two collections agree on all filters F: $X(F) = Y(F)$. For any element $a \in A$, both sets $F = [a)$ and $F_0 = [a) - \{a\}$ are filters, hence $X(a) = X(F) - X(F_0) = Y(F) - Y(F_0) = Y(a)$. □

We illustrate now several cases when $\leq^F_K$ (and, hence, $\leq^\flat_K$ under the conditions of Theorem 6.5) is an order relation. (1) when A is totally unordered: in this case every collection is non-redundant: $X([a)) - X([a) - \{a\}) = X(a) - X(\emptyset) = X(a) - 0 = X(a)$. (2) when K is a cancellative monoid, i.e. $x + z = x + z'$ implies $z = z'$. Here too every collection is non-redundant. This includes the case when K is $\mathbb{N}$, or $\mathbb{R}^+$; in particular, $\leq_{\mathbb{R}^+}$ is an order relation on probabilistic sets. (3) when K is a distributive lattice $(K, \vee, \wedge, 0, 1)$ and forall $a \in A$, and $X(a) \wedge (\bigvee_{b>a} X(b)) = 0$.

We end this section by giving a very simple proof of Theorem 6.5 case (1).

Proof: (of Theorem. 6.5, Case (1): K is a distributive lattice] Let $H_{ba} = y_b$ if $a \leq b$. We verify that this is a lower flow given that $X(F) \leq Y(F)$ for any filter F. We observe that $X([a)) \leq Y([a)) \implies X(a) \leq Y([a))$. We verify the first condition: $\sum_b H_{ba} = \sum_{b \geq a} y_b = Y([a)) \geq x_a$. The second condition $\sum_a H_{ba} = \sum_a y_b = y_b$, where the last equality follows because $+$ is idempotent. □

6.2 Network Flows

We show case (2) of Theorem 6.5 by using an interesting connection to the min-cut max-flow theorem network flows [2]. The classical min-cut max-flow theorem is over $\mathbb{R}^+$, which we use for case (2). There have also been extensions to other semirings, under several restrictions, but to the best of our knowledge

there is no definitive characterization of the semirings on which the min-cut max-flow theorem holds.

Definition 6.7 (Network Flows) *A* flow network $N = (V, E, s, t, c)$ *consists of a directed graph* (V, E)*, two distinguished vertices* s*, the source, and a* t*, the sink, and a function* $c : E \to \mathbb{R}^+$ *called the* capacity function*. A* network flow *is a function* $f : E \to \mathbb{R}^+$ *that satisfies the* Capacity Constraints*,* $\forall e \in E f(e) \leq c(e)$*, and* Conservation*,* $\forall v \in V \quad \sum_{(x,v)\in E} f(x, v) = \sum_{(v,x)\in E} f(v, x)$*. The* value *of a network flow* f *is denoted* $v(f)$ *and is defined by* $v(f) = \sum_{(s,x)\in E} f(s, x)$*. A* cut *is a partition of the vertices* (S, T) *such that* $s \in S$ *and* $t \in T$*. The capacity of a cut* (S, T) *is denoted* $c(S, T)$ *and is defined by*

$$c(S, T) = \sum_{x\in S, y\in T:(x,y)\in E} c(x, y).$$

The min-cut max-flow theorem is:

Theorem 6.8 ([2]) *Consider any flow network* N*, then*

$$\max_{f \text{ is a network flow in } N} v(f) = \min_{(S,T) \text{ is a cut in } N} c(S, T)$$

We show now how case (2) of Theorem 6.5 follows from the min-cut max-flow theorem. In the proof below we make use of the element ∞, thus we need to extend $\mathbb{R}^+$ by adding ∞. This is just a convenience: ∞ can be replaced with any number "big enough"[5].

Proof: (of Theorem 6.5 (2)) Let A' be a disjoint copy of A. For an element $a \in A$ we denote $a' \in A'$ its copy, and similarly $S' \subseteq A'$ denotes the copy of a set $S \subseteq A$. Let X, Y be s.t. $X \leq^F_{\mathbb{R}} Y$. We construct the following flow network $N = (V, E, s, t, c)$.

- $V = A \cup A' \cup \{s, t\}$ for two fresh elements s, t.
- $E = \{(s, a) \mid a \in A\} \cup \{(a, b') \mid a, b \in A, a \leq b\}, \cup \{(b', t) \mid b' \in A\}$
- $c(s, a) = X(a)$, $c(b', t) = Y(b)$, $c(a, b') = \infty$ for $a, b \in A$.

While V is infinite, only a finite number of edges have non-zero capacity.

Recall that $w(X)$ is the weight of X, hence $w(X) = X(A) \leq Y(A) = w(Y)$, because A is a filter. Our proof consists of three steps:

1. If $X \leq^F_{\mathbb{R}^+} Y$ then every cut S, T has capacity $c(S, T) \geq w(X)$; hence there exists a flow f with value $v(f) \geq w(X)$.
2. Let f be a flow with value $w(X)$, and define $g_{ba} = f(a, b)$. This defines an order-compatible matrix $G = (g_{ba})_{a,b}$ satisfying:

$$\forall a, \sum_b g_{ba} = x_a \quad \text{and} \quad \forall b, \sum_a g_{ba} \leq y_b \tag{1}$$

[5] e.g. $\max(w(X), w(Y)) + 1$.

3. If G satisfies Eq.(1), define $H = (h_{ba})_{a,b \in A}$ by $h_{bb} = g_{bb} + (y_b - \sum_a g_{ba})$ and $h_{ba} = g_{ba}$ when $a \neq b$. Then H is a lower homomorphism $H : X \to^\flat Y$.

To prove 1., consider a cut S, T, and assume w.l.o.g. that $c(S,T) < \infty$. Then, if $a \in S$, its copy a' is also in S, otherwise the edge (a, a') gives the cut an infinite capacity. Similarly, if $a \in S$ and $a \leq b$, then $b' \in S$, because $c(a, b') = \infty$. Thus, S and T partition A into a filter F and an ideal I s.t. $S = \{s\} \cup F \cup F'$ and $T = I \cup I' \cup \{t\}$. Thus, $c(S,T) = X(I) + Y(F) = w(X) - X(F) + Y(F) \geq w(X)$.

To prove 2. we note that for any flow f, $v(f) = \sum_a f(s,a) \leq \sum_a c(s,a) = \sum_a X(a) = w(X)$. Hence if $v(f) = w(X)$ we must have $f(s,a) = x_a$ forall a. The left equality in Eq.(1) follows then from the conservation of flow at a node a. Similarly, $f(b', t) \leq c(b', t) = Y(b)$ forall $b \in A$, and the right inequality in Eq.(1) follows from the conservation of flow at the node b'.

Finally, 3. follows from direct calculation.

□

7 Conclusions

We have introduced pre-order relations on collections annotated from a naturally ordered monoid. We studied several properties of these pre-orders: we have shown that under certain conditions they are characterized by the existence of some homomorphisms between the annotated collections, which in turn are characterized by a point-wise order on the annotations on the filters (or ideals) of the collection. We have discussed applications to incomplete databases and to probabilistic databases.

References

[1] P. Ara and E. Pardo. Refinement monoids with weak comparability and applications to regular rings and c # -algebras. *Proceedings of the American Mathematical Society*, 124(3):715–720, March 1996.

[2] Thomas H. Cormen, Charels E Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).

[4] J. N. Foster, T. J. Green, and V. Tannen. Provenance semirings for complex objects and xml., 2007. manuscript.

[5] T. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

[6] T.J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[7] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[8] Leonid Libkin and Limsoon Wong. On representation and querying incomplete information in databases with bags. *Information Processing Letters*, 56:209–214, 1995.

The role of graphs in computer science

Jos Baeten Kees van Hee

Abstract

In this tribute to Jan Paredaens sixties birthday we evaluate the role of graphs in computer science. It turns out that graphs are the essential models for theoretical computer science as well as for software engineering. Special attention is payed to the generations of graphs and the analysis of graphs generated by a probability mechanism. The later topic opens doors for interesting new research.

1 Introduction

Graphs play an important role in many disciplines, as illustrated by some examples: electronic circuits in electrical engineering, bond graphs in mechanical engineering, social networks in social psychology and distribution networks in logistics. Graphs seem to be a natural way to model phenomena, probably because they offer a method to express models by pictures. The foundations of graph formalisms were established by the branch of mathematics called graph theory. However the graph as a modelling concept is so powerful that "scientific users" of graphs did not wait for the graph theorists to extend the graph formalism. So within mathematics there are several other branches, such as combinatorial optimization and the theory of Markov processes, that have contributed to the theory of graphs. Computer science is a young discipline with mathematics and electrical engineering as founding fathers. No wonder that graphs play an essential role in computer science and that computer science has strongly contributed to the extension of graph theory. In computer science the language of mathematics is more used than the millions of mathematical theorems. The mathematical language is used by mathematicians to define generic structures and to prove universally valid properties of these structures. In fact, in mathematics the language is used to communicate mathematical results between mathematicians. The main concern of computer science is the design and construction of computer systems, networks of machines that can process information. Models are used in software engineering as a design for a system and to guide the construction of a system. The language of mathematics is used here to model specific computer systems and to prove properties of these systems. One can say that the language of mathematics is what geometry is for architects. Computer systems may be extremely complex and it is very difficult to model all the details in such a way that one can keep an overview of the

system as a whole. May be this is the reason that computer science has put so much effort in the development of mathematically based frameworks to model computer systems. A *framework* consists of a *language*, defined by a *syntax* and *semantics*, defined by a mapping of the language constructs into a mathematical model. Often a framework has a third component, which is an *inference mechanism* to derive properties of a modelled system. One universally accepted model of a computer system is called an *automaton* or *transition system*. A *transition system* consists of a *state space* and a relation that determines the possible *transitions* from one state to next state. This discrete-event nature is characteristic for computer science. In fact a transition system, expressed with elementary set theory, can be used directly as modelling language for computer systems but this is only a feasible solution for very small systems. So in general we need more sophisticated frameworks having more expressive comfort (i.e. ease to express) than the crude transition system itself. Although transition systems are not suitable modelling complex systems, the frameworks in which we model these systems generate often a (very large) transition system. If such a generated transition system is finite we can apply (computerized) *model checking* techniques to derive properties of the system. There are also techniques to visualize very large transition systems in such a way that human inspection is possible [14]. Transition systems are not always suitable as a semantical model of computer systems since transition systems describe the behavior of a system as a *sequence* of events leading the system from one state to another. So for transition systems the concept of a next state is essential. There are situations in which we can not distinguish a next state and in those cases it is more natural to model a computer system by a *partially ordered* set of events. However in most practical cases the transition system is an adequate semantical model of a system and therefore we adopt it here.

In this paper, which is a tribute to Jan Paredaens, we will discuss the role of graphs as a language for modelling systems. Jan has strongly contributed in the nineties to this development by the the *GOOD-system*, based on the Graph Oriented Object Database model [7] and [6]. In this framework graphs are used to express the type level as well as the instance level of a database in terms of graphs and queries can be expressed in a graphical way as well. Recently Jan has contributed to the modelling of *Jackson nets*, a special class of Petri nets that can be generated by a simple graph grammar [5]. In section 2 we highlight some aspects of the use of graphs to model and analyze computer systems. In section 3 we introduce the Jackson nets because they are a good illustration of graph grammars and we will need them in the last section.

In section 4 we consider *random graphs* and we illustrate this with random Jackson nets. There are at least two reasons to consider random graphs in computer science. The first reason is that there exist many graph models of real computer systems. So we can speak of "populations" of graph models. However it is unclear how to characterize these sets of graph models. Of course one could say that all these models are probably unique. But on the other hand it is unlikely that all these models are purely random. It is comparable with the length of human beings: the probability that two individuals have exactly the

same length is zero, nevertheless the length of persons is normally distributed with a mean of about 180 cm and a spread of about 10 cm. It would be interesting to have a similar notion for a population of graphs. Then it would be possible to identify the probability distribution of graph models in a particular application domain. A second reason for having a probability distribution over a set of graphs is that it allows us to *benchmark* graph algorithms for average performance. The analysis of algorithms for graphs is mainly concerned with worst case behavior. It is hard to say something for the "average" case because it is not clear what the "average" graph is. In practice one uses a (finite) set of representative models that can be considered as a benchmark. If we have a probability distribution for a class of graphs we are able to estimate the average performance of algorithms in a systematic way without using a benchmark set.

2 Graph models for computer systems

Graphs play a very important role in computer science. They are used to model systems as a whole or some aspects of systems. Before we study some of the most used graph frameworks in computer science, we will define the notion of graph. There are several ways to define graphs. We follow here the approach of [5].

Definition 1 (Graph) *A* graph *is defined by a five tuple:*

$$(Node, Edge, \sigma, \tau, \lambda)$$

where Node is a set of nodes or vertices, Edge a set of edges connecting the nodes and $\sigma : Edge \to Node$ *is called the* source *function,* $\tau : Edge \to Node$ *is called the* target *function and* λ *is a* labeling *function with Edge as domain.*

Note that [5] does not consider a labeling function for edges. A graph $G_1 = (N_1, E_1, \sigma_1, \tau_1, \lambda_1)$ is a *subgraph* of a graph $G_2 = (N_2, E_2, \sigma_2, \tau_2, \lambda_2)$ if $N_1 \subseteq N_2$, $E_1 \subseteq E_2$ and $\sigma_1, \tau_1, \lambda_1$ are the appropriate restrictions of $(\sigma_2, \tau_2, \lambda_2)$ respectively.
A *path* in a graph is a alternating sequence of nodes and edges (starting and ending with a node) such that for each $e \in Edge$ in the sequence $\sigma(e)$ is the predecessor of e and $\tau(e)$ the successor of e in the sequence. We say that node e_2 is *reachable* from node n_1 if there is a path starting in n_1 and ending in n_2.

Definition 2 (Typed graph) *A* type graph *is a graph that is used to define a class of graphs. Formally we define such a class by a triple* $(G, type, \tilde{G})$ *where* $\tilde{G} = (\tilde{N}, \tilde{E}, \tilde{\sigma}, \tilde{\tau}, \tilde{\lambda})$ *is a type graph,* $G = (N, E, \sigma, \tau, \lambda)$ *is the graph, called the* typed graph *and type is a* graph morphism *such that type* $= (f_N, f_E)$ *where* $f_N : N \to \tilde{N}$ *and* $f_E : E \to \tilde{E}$ *satisfying:*

$$\forall e \in E : \tilde{\sigma}(f_E(e)) = f_N(s(e)) \wedge \tilde{\tau}(f_E(e)) = f_N(t(e))$$

A typed graph G that belongs in this way to a type graph $\tilde{G}$ *is also called an* instance *of* $\tilde{G}$.

Now we will apply these concepts to define some of the most used modelling frameworks for computer systems.

Definition 3 (Transitionsystem) *A* transition system *is a graph*

$$(State, Event, \sigma, \tau, \lambda)$$

So the nodes are called states *and the edges are called* events. *The labeling function may used to classify the events. The transition relation is defined by*

$$\{(s, \ell, s') | s, s' \in State \wedge \exists e \in Event : \lambda(e) = \ell \wedge \sigma(e) = s \wedge \tau(e) = s'\}$$

In case the labeling function is injective we may identify events and labels and "forget" λ. Note that a Turing machine can be modelled as a transition system with an infinite state space. Transition systems are a good formalism to reason about systems, but they are not suitable for direct modelling complex computer systems. Therefore many dedicated languages have been designed to model complete computer systems or aspects of these systems.

One very important class of graphical modelling frameworks is concerned with modelling of *states* of complex systems. These states are often recorded in a *database* and therefore these models are called *data models*. For expressing data models there are many graphical frameworks. The most classical one is the Entity-Relationship model (ER-model)[3]. Another classical one is the functional datamodel [4]. The first graphical framework that uses graphs not only for the modeling of instances but also for the modeling of queries and updates is the GOOD-model [6].

Definition 4 (Entity-relationship model) *An* ER-model *is in fact a type graph defined by:*

$$\tilde{G} = (\tilde{E}ntity \cup \tilde{R}elationship \cup \tilde{A}ttribute, \tilde{R} \cup \tilde{A}, \tilde{\sigma}, \tilde{\tau}, \tilde{\ell})$$

where all sets are pairwise disjoint and $\sigma : R \to Relationship$, $\sigma : A \to Entity$, $\tau : R \to Entity$ *and* $\tau : A \to Attribute$ *and* λ *is not specified, but it can be used for instance to characterize the cardinality of relationships. Normally the the Entity nodes are rendered as rectangles, the Relationships as diamonds and the Attributes as ovals.*

Definition 5 (Instance of an ER-model) *Let graph* G *be defined by:* $G = (Entity \cup Relationship \cup Attribute, R \cup A, \sigma, \tau, \ell)$ *and let the mappings* (f_E, F_N) *form a graph morphism from* G *to* $\tilde{G}$ *such that* $f_E : Entity \to \tilde{E}ntity$, $f_E : Relationship \to \tilde{R}elationship$ *and* $f_N : R \to \tilde{R}$ *and* $f_N : A \to \tilde{A}$. *Then* G *is an* instance *of* $\tilde{G}$.

G represents a *state* of the system and can be stored in the database of the computer system while $\tilde{G}$ in fact defines the state space of the system. We may say that the semantics of an ER-model $\tilde{G}$ is the set of all instance graphs G. If we consider a database as a (complex) *variable* then the type graph $\tilde{G}$ is its *type*.

An even more important class of graphical modelling languages is concerned with the *behavior* of a computer system. The models in this class are called *process models*. One of the oldest modelling frameworks is the language of *flow charts*. Many graphical modelling languages are developed and used by industry, for example the EPC-language (event-process chains)[10] and BPMN (business process modelling notation). These languages usually suffer from an lack of well-defined semantics. There are two classes of formal frameworks for processes which have good formal semantics in terms of transition systems: the process algebras, like CCS, CSP, ACP and Pi-calculus (cf [1]), and Petri nets, like the classical P/T-nets and the more expressive colored Petri nets [9]. Unlike process algebras, Petri nets have a graphical notation. We give a definition of a classical Petri net here.

Definition 6 (Petri net) *A* Petri net N *is defined as a graph with two kinds of nodes, called places and transitions:*

$$N = (Place \cup Transition, Flow, \sigma, \tau, \lambda)$$

such that $\sigma : Flow \to Place$, $\tau : Flow \to Transition$ *and* $\lambda : Flow \to \{Input, Output\}$, *which indicates if the flow is an input or an output for a transition. Normally, the Places are displayed as circles and the transitions as rectangles or bars.*

In order to see how the graph of a Petri net defines a transition system we have to add the notion of a state, called marking. A *marking* of a Petri net is a mapping $s : P \to \mathbb{N}$. Before we are able to define the transition system we introduce some notation. For $t \in T$ we denote by $\bullet t$ the set of all input places of t, i.e. $\bullet t = \{p \in P | \exists f \in Flow : \sigma(f) = p \wedge \tau(f) = t \wedge \lambda(f) = Input\}$ and similarly $t\bullet = \{p \in P | \exists f \in Flow : \sigma(f) = p \wedge \tau(f) = t \wedge \lambda(f) = Output\}$. It is a good practice to abuse the notation a bit: $\bullet t$ is also used as a function over P with $\bullet t(p) = 1$ if $p \in \bullet t$ and $\bullet t(p) = 0$ otherwise. Similarly we use $t\bullet$ as a function and we apply the usual operators $+$, $-$ and $>$ to functions in the point-wise way .

Definition 7 (Transition system of a Petri net) *The* transition system *of the Petri net is denoted by* $(\tilde{State}, \tilde{Event}, \tilde{\sigma}, \tilde{\tau}, \tilde{\lambda})$ *and it is defined by:*

$$\tilde{State} = P \to \mathbb{N} \wedge \tilde{Event} = \{(s, t, s') | s, s' \in \tilde{State} \wedge t \in T \wedge s' = s - \bullet t + t\bullet \wedge s \geq \bullet t\}$$

and for $e \in \tilde{Event}$ *denoted by* $e = (s, t, s')$ *we have:*

$$\tilde{\sigma}(e) = s, \tilde{\tau}(e) = s', \tilde{\lambda}(e) = t$$

So a state is a marking. The set of events is constructed in such a way that a transition t can "fire" only if all its input places have enough tokens. The effect of the firing of a transition t is that from all input places of t one token is subtracted and for all output places a token is added. Given one initial state

(marking) the subgraph of the transition system that contains all reachable states is called the *reachability graph* of the Petri net.

Now we have seen that graph languages offer a powerful tool for expressing models in computer science. However we have not seen how they can be used in the *modelling process*. How do we develop graphical models?. There are two famous approaches in computer science: *top down* design, also called *stepwise refinement* and *bottom up* design. If we apply this to graphs as our modelling objects, then in the top down approach we will start with a simple graph and we will refine it. In the bottom up approach we start with many graphs and we will glue them together. In both cases we have to deal with *graph transformations*. Graph transformations can be defined by *production rules*. Here we follow the approach of [5] in a simplified form. A *production rule* is a triple (L, K, R) where L, K and R are all graphs and K is common subgraph of L and R. The first step in the application of a production rule is that in a graph G the graph L is *detected*, i.e. there is found a subgraph of G that is isomorphic with L. The second step is that the graph G is transformed into $\tilde{G} = G\backslash(L\backslash K) + (R\backslash K)$. The graph K is the *interface* between G and L and R, i.e. $G\backslash L$ and $R\backslash K$ are only connected via K. (Here we use $\backslash$ for graph subtraction and $+$ for graph addition.) There are some "sanity" requirements for these operations such that all intermediate results are proper graphs. A *graph transformation system* is a set of production rules and a *graph transformation system* together with an initial graph is called a *graph grammar*. Instead of the graphs L, K and R in production rules we often use *patterns*, which are specifications for a set of graphs. So we do not have to have a perfect match of a L with a subgraph of G but we have to find a subgraph of G that fits into the pattern of L. We could think of patterns as type graphs, however we will not elaborate the notion of pattern here.
With a graph grammar we may define a class of graphs: all graphs that can be derived by applying the production rules in a finite number of steps. It is possible to generate only graphs that belong to a specific type. This is, as we have seen, important for data models. In the graphical data modelling framework GOOD the queries and updates of an instance are specified by production rules.

3 A graph grammar for a class of Petri nets

In this section we introduce a class of Petri nets called Jackson nets. For a detailed description and analysis of Jackson nets see [13]. First we introduce workflow nets, a special class of Petri nets, because Jackson nets are a subclass of workflow nets.

Definition 8 (Workflow net) *A workflow net is a Petri net with one source place i and one sink place o, i.e. i has no input transitions and o has no output transitions. Further any node x (which is either a place or a transition) is on a directed path from the source place i to the sink place o.*

Workflow nets (cf [12]), as the name suggest, are used to model business procedures as we encounter in organizations and in their supporting information systems.

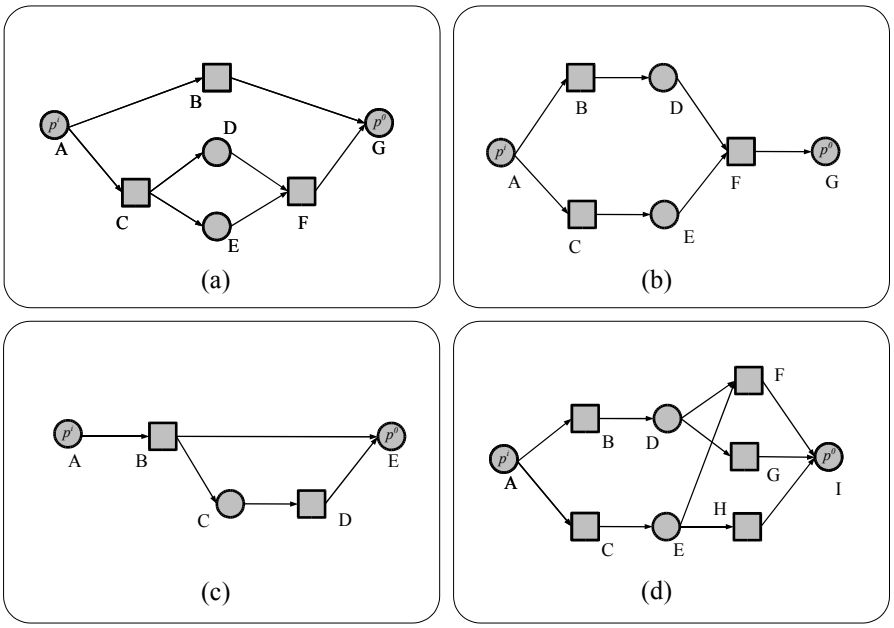


Figure 1: Examples of workflow nets

A "sanity check" for business procedures is the possibility of proper termination. This property is for workflow nets is called *soundness*.

Definition 9 (Soundness) *A workflow net is called* k-sound*, for $k \in \mathbb{N}$ if for each marking m that is reachable from an initial marking with only k tokens in the source place i, the final marking with only k tokens in the sink place o can be reached. A workflow net is called* ω -sound *if it is k-sound for all $k \geq 1$.*

Note that "1-sound" is usually called "sound". In Figure 1 only net (a) is ω-sound, the others are not k-sound for any k.
Next, we give five rules R1,..., R5, displayed Figure 2 to generate nets starting with a net with only one place. Rule $R1$ replaces a place p_1 by two places p_2 and p_3 with a transition t_1 in between. All input transitions of p_1 become input transition for p_2 and all output transitions of p_1 become the output transitions of p_3. So $\bullet p_1 = \bullet p_2$ and $p_1\bullet = p_3\bullet$. In terms of graph grammars $R1$ is set of production rules (L_1, K_1, R_1) where L_1 is a sub-net with only one place p_1 and $\bullet p_1 \cup p_1\bullet$ as transitions. The interface K is the sub-net with no places and $\bullet p_1 \cup p_1\bullet$ as transitions and R_1 is the net with two places $\{p_2, p_3\}$ and

as transitions $\bullet p_1 \cup \{t_1\} \cup p_1\bullet$. Rule $R2$ is similar to $R1$ but now the role of places and transitions are exchanged. Rule $R3$ is adding a transition t_1 to a an arbitrary place p_1. Rules $R4$ and $R5$ duplicate a place p_1 and a transition t_1 respectively. Duplication means for $R4$ that place p_1 is replaced by two places p_2 and p_3 such that $p_1\bullet = p_2\bullet = p_3\bullet$ and $\bullet p_1 = \bullet p_2 = \bullet p_3$. Rule $R5$ is similar.

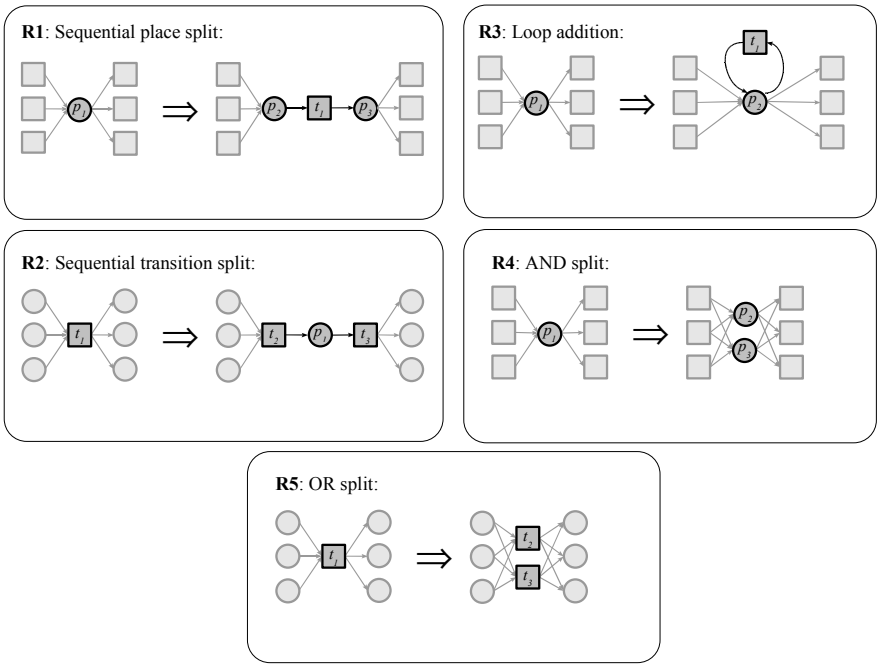


Figure 2: The generation rules for Jackson nets

These rules were first studied by Berthelot in [2] and Murata in [11] as *reduction* rules that preserve liveness and boundedness properties of Petri nets. The rules are often called the "Murata rules". In fact Murata considers one rule more, a loop addition with a (marked) place, similar to R3. We do not use this rule since it would destroy the soundness property: since the place added to the transition should contain always one token.

Definition 10 (Jackson Net) *A* Jackson net N *is a net that can be generated, from one place, by applying the rules* $R1, .., R5$ *recursively, however rule* $R3$ *should not be applied in the first step and never to the source and sink place of a net.*

Remark that the net of Figure 1 (a) is a Jackson net. Its generation is given in Figure 3.

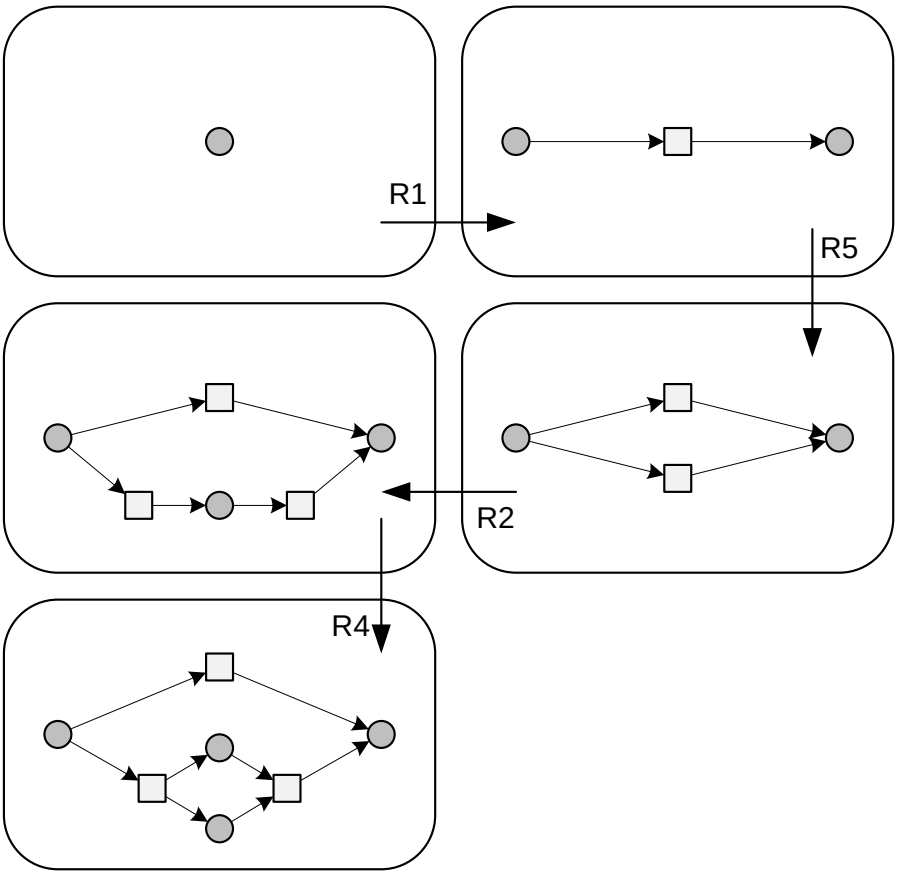


Figure 3: Generation of a net

It is well-known that the Murata rules preserve liveness and boundedness of Petri nets (see [11]) with respect to a given marking. In [12] it is shown that 1-soundness is equivalent with liveness and boundedness of the *closure* of a workflow net, i.e. the Petri net obtained from a workflow net by adding one transition $t^\star$ that connects the sink place o to the source place i, in the initial marking with only one token in i.
In [[13] it is proven that all Jackson nets are ω-sound. There it is also shown that Jackson nets can be generated by *process algebraic* expressions with four operators: sequence, choice, parallel and iteration.

The class of Jackson nets contains interesting sub-classes. One of them is the class of *flow charts*, which are in fact sequential processes or state machines. This class is obtained by starting with one place and using only the rules $R1$, $R2$, $R3$ and $R4$, so rule $R5$ is left out. Another class, called *structured Jackson nets* is defined by the rules $R6, ..., R9$ displayed in Fig. 4. These rules only refine

transitions. It is easy to verify that these new rules can be constructed from the old rules: $R6 = R2$, $R7 = R2 + R1 + R5$, $R8 = R2 + R4 + 2 \times R1$ and $R9 = R2 + R3 + R1 + R2$.
We did not address the problem of *parsing* a given graph, but we remark that bottom up parsing by applying the rules in opposite direction is the obvious approach. Only if we succeed in reducing the graph to one place, the graph did belong to the grammar.

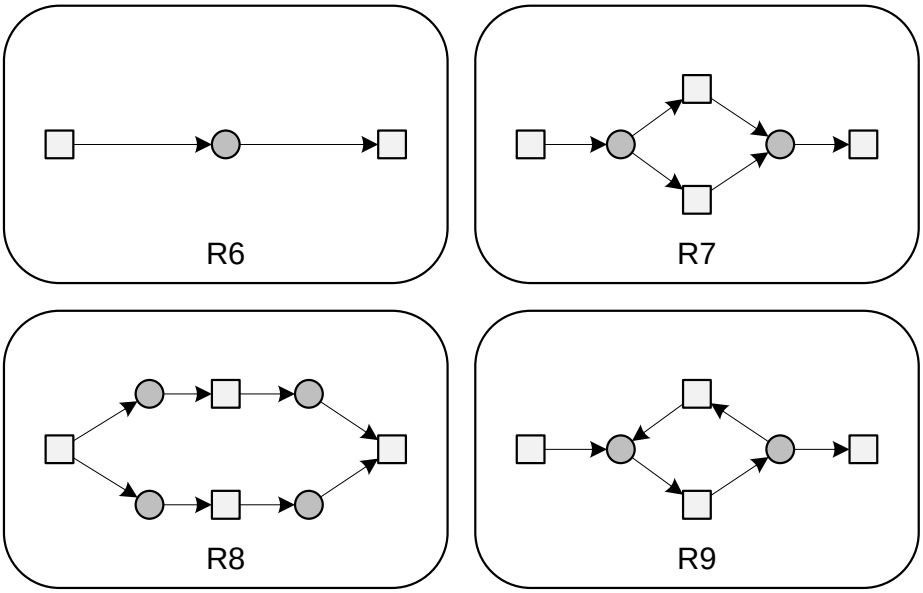


Figure 4: Rules for structured Jackson nets

4 Random graphs

As motivated in the introduction it is interesting to have probability distributions over graphs. If a class of graphs is finite then it is easy to define a probability distribution, for instance the *uniform* distribution that gives all graphs the same probability will do. If the class is infinite it is less obvious to define a probability distribution because the total probability mass should be one. We propose here several methods to define probability distributions for infinite classes of graphs generated by a graph grammar. First we recall some facts from probability theory.
We need some discrete probability distributions on $\mathbb{N}$ ($0 \in \mathbb{N}$). Here N is a random variable and as usual $\mathbb{P}$ is the probability density, $\mathbb{E}$ is the expectation and σ^2 is the variance.

1. *Bernoulli* distribution $\mathbb{P}[N = 1] = p \wedge \mathbb{P}[N = 0] = 1 - p$ with $\mathbb{E}[N] = p$ and $\sigma^2[N] = p(1 - p)$
2. *Binomial* distribution $n \in \{0, ...k\} : \mathbb{P}[N = n] = \binom{k}{n} p^n (1 - p)^{n-k}$ with $\mathbb{E}[N] = kp$ and $\sigma^2[N] = kp(1 - p)$
3. *Geometric* distribution $\forall n \in \mathbb{N} : \mathbb{P}_s[N = n] = p^n(1 - p)$ with $\mathbb{E}[N] = p/(1 - p)$ and $\sigma^2[N] = p/(1 - p)^2$
4. *Negative binomial* distribution $\forall n \in \mathbb{N} : \mathbb{P}_s[N = n] = \binom{n+k-1}{k-1} p^n (1 - p)^k$ with $\mathbb{E}[N] = kp/(1 - p)$ and $\sigma^2[N] = kp/(1 - p)^2$
5. *Poisson* distribution with for $\forall n \in \mathbb{N} : \mathbb{P}[N = n] = \frac{\lambda^n}{n!} e^{-\lambda}$ and $\mathbb{E}[N] = \lambda$ and $\sigma^2[N] = \lambda$

Note that these distributions are closely related. So has the sum of k identically and independent distributed (iid) Bernoulli random variables has a binomial distribution. The first 0 in a sequence of iid Bernoulli random variables has a geometric distribution. The sum of k iid geometrical distributed random variables has a negative binomial distribution and finally the binomial and the negative binomial distribution converge to a Poisson distribution if $k \to \infty$ and $p \to 0$ such that $kp \to \lambda$.

Let a graph grammar Γ be given with a finite set of production rules $\Re$. Define an arbitrary probability distribution $\mathbb{P}_r$ over the set of rules $\Re$:

$$\forall r \in \Re : \mathbb{P}_r(r) \geq 0 \wedge \sum_{r \in \Re} \mathbb{P}_r(r) = 1$$

For some of the algorithms we need a *stopping time* N , i.e. a random variable with a probability distribution $\mathbb{P}_s$ for the total number of times we apply some production rule.

We present the methods in terms of algorithms to generate graphs. In the first two algorithms we assume that we are able to detect all patterns in a give graph where the left-hand side of a rule is applicable and then we select one at random, i.e. with the same probability. We denote by F the function that transforms a graph: $F(G, r, \ell)$ is the graph obtained from G where rule r is applied at subgraph ℓ, so the left-hand side of r matched with subgraph ℓ of G.

Algorithm 1: Geometric stopping rule

```
start with an initial graph G;
set i := 1;
while i=1 do
    generate a rule r (from P_r);
    generate a random subgraph ℓ of G where r is applicable;
    set G := F(G, r, ℓ);
    generate a new value for i (either 0 or 1) from a Bernoulli distribution;
end
```

In the first method the number of steps is geometric distributed while in the next method there is an arbitrary stopping distribution.

Algorithm 2: General stopping rule

start with an initial graph G;
generate the stopping time n (from $\mathbb{P}_s$);
set $i := 0$;
while $i < n$ **do**
  generate a rule r (from $\mathbb{P}_r$);
  generate a random subgraph ℓ of G where r is applicable;
  set $G := F(G, r, \ell)$;
  set $i := i + 1$;
end

Note that both algorithms can be applied to the generation of Jackson nets, as discussed in the preceding section. For the next method we assume more structure and therefore we will assume now that we are dealing with a graph grammar where only one type of node can be refined. The structured Jackson nets are an example of this. In this method we will label the nodes (places and transitions) during the generation process. No rule will be applied to a labeled node. Each unlabeled node will be selected at some point in time, not necessarily at random. If a node is selected a Bernoulli trial is performed and if the outcome is 0 the node will be marked (and not further refined), otherwise a random selected rule will be applied. We start with a Jackson net J with only one transition t connected to source and a sink place. Node t is unlabelled.

Algorithm 3: For structured Jackson nets

set $G := J$;
while *there are unmarked nodes* **do**
  select an arbitrary unlabelled node ℓ;
  generate a Bernoulli variable i;
  if $i = 0$ **then**
    mark the node;
  else
    generate according to $\mathbb{P}_r$ a rule for ℓ;
    $G := F(G, r, \ell)$;
  end
end

In order to identify a population of graphs generated in this way we will compute some *characteristic* values. Typical examples are the expected *number of nodes* (of a certain type), or more general the expected *number of subgraphs* of a special structure, the expected *length of the shortest path* from a source node to a sink node (if they are defined for the population). Other characteristics may be the expected *fan out* and *fan in* of a node. Often these values are difficult to compute in an analytical way, then Monte Carlo simulation offers a solution. For the class of Jackson nets we will compute some of these characteristics. Let

the probabilities of the rules $R1, ..., R5$ be $p_1, ..., p_5$ respectively. In each step i of the algorithms let X_i be the random rule, so for $j \in \{1, ..., 5\} : \mathbb{P}[X_i = Rj] = p_j$ with $\sum_{j=1}^{5} p_j = 1$. Similarly we have for the rules $R6, ..., R9$ the probabilities $p_6, ..., p_9$ with $\sum_{j=6}^{9} p_j = 1$. Note that X_i and N are independent.

- Consider algorithm 1 for a Jackson net nets with that start with J. The expected *number of nodes* is derived using Wald's formula:

$$\mathbb{E}[\sum_{i=1}^{N} X_i] = \mathbb{E}[N]\mathbb{E}[X_i] = \mathbb{E}[N](2(p_1 + p_2) + (p_3 + p_4 + p_5))$$

  since the first two rules generate two extra nodes and the others only one. So in case we have a negative binomial distribution for N and all rules have the same distribution the expected number is $\frac{9}{5}kp/(1-p)$.

- In case of algorithm 2 we have the same formula but with $k = 1$

- Same situation but now we are computing the expected number of places and transitions separately. Note that rules $R1$, $R2$ and $R4$ produce one extra pace and the others none. Hence for the places we have $\mathbb{E}[X_i] = (p_1 + p_2 + p_4)$ and for the transitions we obtain similarly $\mathbb{E}[X_i] = (p_1 + p_2 + p_3 + p_5)$

- For algorithm 3, applied to structured Jackson nets the expected number of transitions V is expressed by the recursive equation $V = p(\mathbb{E}[X_i] + V)$ which results in $V = \mathbb{E}[X_i]/(1-p)$, where $\mathbb{E}[X_i] = p_6 + 3(p_7 + p_8 + p_9)$. This exactly the same result as if we apply algorithm 2 to these nets.

An interesting subclass of the random Jackson nets is called *series-parallel graphs* as studied in [8].This subclass is generated from J by an algorithm similar to algorithm 3, using only rules $R2$ and $R5$ (only transition refinement), where we set $p_2 = q$ and $p_5 = 1 - q$. They consider the situation that the process continues for ever, so the Bernoulli parameter $p = 1$. (In [8] it is assumed that a rule is applied to all transitions in one *step*.) Of course the expected number of nodes is infinite, but there are some interesting results. They show that the value $q = 1/2$ is critical in the sense that the random variable that represents the *distance* from the source to the sink becomes infinite with probability one if $q > 1/2$ and that it tends to proper distribution for $q < 1/2$. We show here two other properties. Note that there is always one source place and one sink place for the whole net. We need the notions of a separating node and a cluster. A *separating* node (place or transition) separates the whole net in two disjoint sub-nets such that each path from a node in one part to a node in the other part should pass by the separating node. A *cluster* is a sub-net with one source place and one sink place that are separating nodes. See Fig. 5

Remark that as soon as $R5$ is applied to a transition that is a separating node, a cluster has been formed. In the first step we either apply $R5$, which means that all generated nets form only one cluster, or we have two nets of the form J that behave independently an identically to the starting net J.We may use this property to compute the estimated number of clusters.

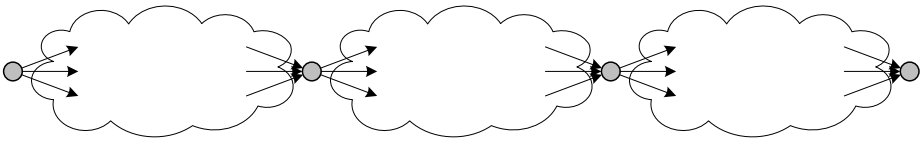


Figure 5: Three clusters

- The expected *number of clusters* V for series-parallel graphs satisfies the equation $V = (1-q) + 2qV$ so V becomes:

$$V = \frac{1-q}{1-2q}$$

  Hence for $q < 1/2$ we have a finite number of clusters although the graph continues splitting for ever!

- The expected *number of source-sink transitions*that connect the source of the whole net with the sink of it, is determined as follows. Assume Y_n is the random variable that indicates the number of these source-sink connecting transitions in step n. Note that in one step all these transitions either produce another source-sink transition or they loose this property. Then

$$\mathbb{E}[Y_{n+1}|Y_n = m] = \sum_{i=0}^{m} 2i \binom{m}{i} (1-q)^i q^{m-i} = 2m(1-q)$$

  So $\mathbb{E}[Y_{n+1}|Y_n] = 2(1-q)Y_n$ hence $\mathbb{E}[Y_{n+1}] = 2(1-q)\mathbb{E}[Y_n]$ and therefore $\mathbb{E}[Y_n] = (2(1-q))^n\mathbb{E}[Y_0] = (2(1-q))^n$. Hence if $q > 1/2$ we have that $\mathbb{E}[Y_n]$ tends to 0 if $n \to \infty$. So in that case there are almost surely no source-sink transitions. In case $q < 1/2$ then $\mathbb{E}[Y_n]$ tends to ∞ if $n \to \infty$, which implies that there is for each n a positive probability that there are at least n source-sink transitions.

5 Future work

At least two topics are interesting to explore further. One is concerned with the use of graphs in modelling computers systems. Although there are good graphical frameworks for modelling state spaces of systems (data models) and there are good frameworks for modelling system behavior (process models) there is to our knowledge still no integrated graphical framework that is good for modelling both aspects of a system.
The other topic is concerned with the random graphs. There are many open questions: For instance what is the expected length of a random Jackson net if all production rules are applied? The identification of graph distributions given

a population is an open problem. The use of these distributions for analysis of algorithms is still open. Although we illustrated the use of random graphs for Petri nets, it might be interesting as well for databases. There it could be used to determine the performance of updates and queries. GOOD would be a good candidate for such an analysis.

References

[1] J. Baeten and W. Weijland. *Process algebra*. Cambridge Tracts in Theoretical Computer Science no. 18. Cambridge University Press, 1990.

[2] G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987.

[3] P. Chen. The Entity-Relationship Model: Towards a unified view on data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[4] D. Shipman. The functional data model and the data language dataplex. *ACM Transactions on Database Systems*, 6(1):140–173, 1981.

[5] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformations*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 2006.

[6] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database end-user interfaces. *ACM: SIGMOD Int. Conf. on Management of Data*, 19(2), 1990.

[7] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. *ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 1990.

[8] B. Hambly and J. Jordan. A random hierachical lattice: the series-parallel graph and its properties. *Advances in Applied Probability*, 36(3).

[9] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.

[10] G. Keller, M. Nüttgens, and A. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[11] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

[12] W. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[13] K. van Hee, J. Hidders, G. Houben, J. Paredaens, and P. Thiran. On the relationship between workflow models and document types. Report CSR 16-07, Eindhoven University of Technology, submitted for publication, 2007.

[14] H. Verbeek, A. Pretorius, W. van der Aalst, and J. van Wijk. On petri-net synthesis and attribute-based visualization. In D. Moldt, F. Kordon, K. van Hee, J.-M. Colom, and R. Bastide, editors, *Proceedings of Workshop on Petri Nets and Software Engineering (PNSE'07)*, pages 127–141, Siedlce, Poland, June 2007. Publishing House of University of Podlasie.

A Complete Axiomatization for Core XPath 1.0

Balder ten Cate, Tadeusz Litak, Maarten Marx

Abstract

This paper provides a complete algebraic axiomatization of node and path equivalences in Core XPath 1.0. Our completeness proof builds on a completeness result of Blackburn et al. [3] for a modal logic of finite trees.

This paper is dedicated to Jan Paredaens on the occasion of his 60th birthday. We are grateful to Paredaens for raising several generations of excellent researchers fond of foundational issues!

The authors of this paper were all trained as modal logicians. Completeness proofs and models which look like trees are at the core of modal logic. So we took an existing result and looked at it as we think Jan Paredaens would have done: "What can this teach me about XML and XPath?" We hope that Jan will like the outcome, which consists of a small gem, a surprise and an open problem.

1 Introduction

In this paper, we consider the problem of finding complete axiomatizations for fragments of XPath. By an axiomatization we mean a set of (valid) equivalences between XPath expressions plus a number of inference rules extending those of equational logic. Completeness means that any two equivalent expressions can be rewritten to each other using just the given equivalences and rules. Completeness tells us, in a mathematically precise way, that the given equivalences capture everything there is to say about semantic equivalence of XPath expressions.

We are aware of only two such results. The first one is the axiomatization of the downward, positive and filter-free fragment of XPath [1]. The other result [16] axiomatizes Core XPath 2.0 which is a very large fragment, with non-elementary complexity for query containment.

In this paper, we axiomatize Core XPath 1.0, which was introduced in [4, 5]. We will be concerned with several notions of equivalence over XML models:

1. root-equivalence of path expressions,
2. strong equivalence of path expressions,
3. (strong) equivalence of node expressions.

We will reduce the problem of axiomatizing these equivalences to a similar problem for a subclass of the Core XPath 1.0 node equivalences introduced by Blackburn et al. [3], which we call *simple node expressions*. The framework of Blackburn et al. [3] at first sight may seem different than ours. It is logically, not algebraically oriented—and so is their notation. However, the translations between their *LOFT formulas* and our *simple node expressions* turn out to be fairly straightforward. A minor difference is that in their framework there is no assumption that every node has exactly one label. This assumption, natural as it is in the XML setting, is not so common for other application of finite trees in, e.g., linguistics (which was the original motivation for Blackburn et al. [3]) or even in computer science itself. Nevertheless, we show it is possible to completely axiomatize both uni-label and multi-label trees in an unified way.

2 Preliminaries

2.1 Syntax, Semantics, Consequence Relations

Let Σ be an infinite set of node-labels. The syntax of Core XPath 1.0 is defined as follows:

$$\begin{aligned}
\mathsf{Axis} &:= . \mid \downarrow \mid \leftarrow \mid \uparrow \mid \rightarrow \mid \downarrow^+ \mid \leftarrow^+ \mid \uparrow^+ \mid \rightarrow^+ \\
\mathsf{PathExpr} &:= \mathsf{Axis} \mid \mathsf{PathExpr}[\mathsf{NodeExpr}] \mid \mathsf{PathExpr}/\mathsf{PathExpr} \mid \\
&\quad\ \mathsf{PathExpr} \cup \mathsf{PathExpr} \\
\mathsf{NodeExpr} &:= v \mid \langle \mathsf{PathExpr} \rangle \mid \neg \mathsf{NodeExpr} \mid \mathsf{NodeExpr} \vee \mathsf{NodeExpr} \ \ (v \in \Sigma)
\end{aligned}$$

We use the following syntactic conventions: v, v' range over elements of the set Σ of *labels*, Greek letters $\phi, \psi \ldots$ over elements of $\mathsf{NodeExpr}$, and Roman capitals A, B, C over elements of $\mathsf{PathExpr}$.

Note that we include the non-transitive sibling axes $\rightarrow$ and $\leftarrow$ in the language. Also, we use angled brackets to distinguish path expressions from node expressions that test for the existence of a path.

Our models are Σ-labeled finite sibling-ordered trees. If every node has exactly one label, we say the model is *an XML model*. However we do not require that all models are XML models. Therefore, models in general are called *multi-label models*. The semantics of Core XPath 1.0 is defined in Table 1 by the functions $[\![\cdot]\!]_{\mathsf{PExpr}}$ and $[\![\cdot]\!]_{\mathsf{NExpr}}$ which take a model and a (path or node) expression as argument. The model is kept implicit in our notation.

Definition 1 (Equivalence Relations)**.** *Let $\phi, \psi \in \mathsf{NodeExpr}$, $A, B \in \mathsf{PathExpr}$. We say that*

- *ϕ is* semantically equivalent *to ψ (notation $\vDash_{\mathsf{xml}} \phi \equiv \psi$) if for arbitrary XML model $[\![\phi]\!]_{\mathsf{NExpr}} = [\![\psi]\!]_{\mathsf{NExpr}}$.*
- *A is* semantically (strongly) equivalent *to B (notation $\vDash_{\mathsf{xml}} A \equiv B$) if for arbitrary XML model $[\![A]\!]_{\mathsf{PExpr}} = [\![B]\!]_{\mathsf{PExpr}}$.*

Table 1: Semantics of Core XPath 1.0.

$$
\begin{array}{lcl}
[\![\mathsf{Axis}]\!]_{\mathsf{PExpr}} & := & \{(x,y) \mid x\mathsf{Axis}y \text{ holds in the tree}\} \\
[\![A/B]\!]_{\mathsf{PExpr}} & := & [\![A]\!]_{\mathsf{PExpr}}/[\![B]\!]_{\mathsf{PExpr}} \\
[\![A \cup B]\!]_{\mathsf{PExpr}} & := & [\![A]\!]_{\mathsf{PExpr}} \cup [\![B]\!]_{\mathsf{PExpr}} \\
[\![A[\phi]]\!]_{\mathsf{PExpr}} & := & \{(x,y) \mid (x,y) \in [\![A]\!]_{\mathsf{PExpr}} \text{ and } y \in [\![\phi]\!]_{\mathsf{NExpr}}\} \\
 & & \\
[\![v]\!]_{\mathsf{NExpr}} & := & \{x \mid x \text{ is labelled with } v\} \\
[\![\langle \mathsf{PathExpr} \rangle]\!]_{\mathsf{NExpr}} & := & \{x \mid \exists y.(x,y) \in [\![\mathsf{PathExpr}]\!]_{\mathsf{PExpr}}\} \\
[\![\neg\phi]\!]_{\mathsf{NExpr}} & := & \{x \mid x \notin [\![\phi]\!]_{\mathsf{NExpr}}\} \\
[\![\phi_1 \vee \phi_2]\!]_{\mathsf{NExpr}} & := & [\![\phi_1]\!]_{\mathsf{NExpr}} \cup [\![\phi_2]\!]_{\mathsf{NExpr}}
\end{array}
$$

- *A is* semantically root-equivalent *to B (notation $\vDash_{\mathsf{xml}} A \equiv^r B$) if for arbitrary XML model with root r and arbitrary element of the universe x, $(r,x) \in [\![A]\!]_{\mathsf{PExpr}}$ iff $(r,x) \in [\![B]\!]_{\mathsf{PExpr}}$.*

Equivalences over multi-label models are defined analogously. We prefix them with $\vDash_{\mathsf{ml}}$ instead of $\vDash_{\mathsf{xml}}$.

These three notions can be reduced to each other, and we will use these reductions in our completeness proofs. To state them elegantly, we develop some extra syntactic machinery.

The *converse* of an axis is defined in the obvious way. The definition is extended inductively to all path expressions as follows:

$$
\begin{aligned}
(A[\phi])^{-1} &:= .[\phi]/A^{-1} \\
(A/B)^{-1} &:= B^{-1}/A^{-1} \\
(A \cup B)^{-1} &:= A^{-1} \cup B^{-1}
\end{aligned}
$$

Lemma 2. *For any model, $(x,y) \in [\![A]\!]_{\mathsf{PExpr}}$ iff $(y,x) \in [\![A^{-1}]\!]_{\mathsf{PExpr}}$.*

Proof. Direct verification. □

We use the following abbreviations:

$A \subseteq B$	for	$A \cup B \equiv B$	$\phi \leq \psi$	for	$\phi \vee \psi \equiv \psi$
true	for	$\langle . \rangle$	false	for	$\neg\mathsf{true}$
$\bot$	for	$.[\mathsf{false}]$	$\phi \wedge \psi$	for	$\neg(\neg\phi \vee \neg\psi)$
leaf	for	$\neg\langle\downarrow\rangle$	root	for	$\neg\langle\uparrow\rangle$
first	for	$\neg\langle\leftarrow\rangle$	last	for	$\neg\langle\rightarrow\rangle$
a^*	for	$. \cup \mathsf{a}^+$	$\langle \mathsf{a}^*[\phi] \rangle$	for	$\phi \vee \langle \mathsf{a}^+[\phi] \rangle$
$\mathsf{E}\phi$	for	$\langle\uparrow^*[\langle\downarrow^*[\phi]\rangle]\rangle$	$\mathsf{A}\phi$	for	$\neg\mathsf{E}\neg\phi$

Lemma 3. *For arbitrary model*

- $[\![\mathsf{root}]\!]_{\mathsf{NExpr}} = \{x \mid \forall y.(x,y) \notin [\![\uparrow]\!]_{\mathsf{PExpr}}\}$
- $[\![\mathsf{leaf}]\!]_{\mathsf{NExpr}} = \{x \mid \forall y.(x,y) \notin [\![\downarrow]\!]_{\mathsf{PExpr}}\}$
- $[\![\mathsf{first}]\!]_{\mathsf{NExpr}} = \{x \mid \forall y.(x,y) \notin [\![\leftarrow]\!]_{\mathsf{PExpr}}\}$
- $[\![\mathsf{last}]\!]_{\mathsf{NExpr}} = \{x \mid \forall y.(x,y) \notin [\![\rightarrow]\!]_{\mathsf{PExpr}}\}$
- $[\![\mathsf{a}^*]\!]_{\mathsf{PExpr}} = [\![.]\!]_{\mathsf{PExpr}} \cup [\![\mathsf{a}^+]\!]_{\mathsf{PExpr}}$
- $[\![\mathsf{E}\phi]\!]_{\mathsf{NExpr}} = \emptyset$ *iff* $[\![\phi]\!]_{\mathsf{NExpr}} = \emptyset$. *Otherwise,* $[\![\mathsf{E}\phi]\!]_{\mathsf{NExpr}}$ *is the set of all nodes.*

Proof. Direct verification. □

Now we can exactly characterize the relationships between different notions of consequence. We need an auxiliary notion for this purpose.

Definition 4. *For arbitrary list of path or node expressions* $\overline{P}$*, let* $\Sigma(\overline{P})$ *be the set of all elements of* Σ *which appear in* $\overline{P}$*. Let*

$$\mathsf{lab}_{\overline{P}} := \bigwedge_{v \neq v'}^{v,v' \in \Sigma(\overline{P})} \mathsf{A}(\neg v \vee \neg v')$$

Lemma 5. *For arbitrary model and arbitrary list of path or node expressions* $\overline{P}$*,* $[\![\mathsf{lab}_{\overline{P}}]\!]_{\mathsf{NExpr}} = \emptyset$ *iff there is at least one pair of distinct labels* $v, v' \in \Sigma(\overline{P})$ *with non-disjoint interpretation, i.e.,* $[\![v]\!]_{\mathsf{NExpr}} \cap [\![v']\!]_{\mathsf{NExpr}} \neq \emptyset$*. Otherwise, i.e., if interpretations of all labels are disjoint,* $[\![\mathsf{lab}_{\overline{P}}]\!]_{\mathsf{NExpr}}$ *is equal to the whole set of nodes.*

Proof. Follows from Lemma 3. □

Corollary 6. *For arbitrary XML model and arbitrary list of path or node expressions* $\overline{P}$*,* $[\![\mathsf{lab}_{\overline{P}}]\!]_{\mathsf{NExpr}}$ *is equal to the whole set of nodes. Hence for arbitrary* $A \in \mathsf{PathExpr}$ *and* $\phi \in \mathsf{NodeExpr}$*,* $\models_{\mathsf{xml}} A[\mathsf{lab}_{\overline{P}}] \equiv A$ *and* $\models_{\mathsf{xml}} \phi \wedge \mathsf{lab}_{\overline{P}} \equiv \phi$

Proof. Follows from Lemma 5: in XML models, denotations of all labels are disjoint. □

Lemma 7. *For every* $A, B \in \mathsf{PathExpr}$*,* $\phi, \psi \in \mathsf{NodeExpr}$*.*

1. $\models_{\mathsf{xml}} A \equiv^r B$ *iff* $\models_{\mathsf{xml}} \langle A^{-1}[\mathsf{root}]\rangle \equiv \langle B^{-1}[\mathsf{root}]\rangle$*.*
2. $\models_{\mathsf{ml}} A \equiv^r B$ *iff* $\models_{\mathsf{ml}} \langle A^{-1}[\mathsf{root}]\rangle \equiv \langle B^{-1}[\mathsf{root}]\rangle$
3. $\models_{\mathsf{xml}} A \equiv^r B$ *iff* $\models_{\mathsf{xml}} .[\mathsf{root}]/A \equiv .[\mathsf{root}]/B$
4. $\models_{\mathsf{ml}} A \equiv^r B$ *iff* $\models_{\mathsf{ml}} .[\mathsf{root}]/A \equiv .[\mathsf{root}]/B$
5. $\models_{\mathsf{ml}} A \equiv B$ *implies* $\models_{\mathsf{ml}} \langle A\rangle \equiv \langle B\rangle$
6. $\models_{\mathsf{xml}} A \equiv B$ *implies* $\models_{\mathsf{xml}} \langle A\rangle \equiv \langle B\rangle$*.*

7. $\vDash_{\mathsf{ml}} A \equiv B$ *iff* $\vDash_{\mathsf{ml}} \langle A[v]\rangle \equiv \langle A[v]\rangle$ *for arbitrarily chosen* $v \notin \Sigma(A,B)$.

8. $\vDash_{\mathsf{ml}} A \equiv B$ *iff* $\vDash_{\mathsf{ml}} \downarrow^*[v]/A \equiv^r \downarrow^*[v]/A$.

9. $\vDash_{\mathsf{ml}} A \equiv B$ *implies* $\vDash_{\mathsf{xml}} A \equiv B$

10. $\vDash_{\mathsf{ml}} A \equiv^r B$ *implies* $\vDash_{\mathsf{xml}} A \equiv^r B$

11. $\vDash_{\mathsf{ml}} \phi \equiv \psi$ *implies* $\vDash_{\mathsf{xml}} \phi \equiv \psi$

12. $\vDash_{\mathsf{xml}} A \equiv B$ *iff* $\vDash_{\mathsf{ml}} A[\mathsf{lab}_{A,B}] \equiv B[\mathsf{lab}_{A,B}]$,

13. $\vDash_{\mathsf{xml}} A \equiv^r B$ *iff* $\vDash_{\mathsf{ml}} A[\mathsf{lab}_{A,B}] \equiv^r B[\mathsf{lab}_{A,B}]$,

14. $\vDash_{\mathsf{xml}} \phi \equiv \psi$ *iff* $\vDash_{\mathsf{ml}} \phi \wedge \mathsf{lab}_{\phi,\psi} \equiv \psi \wedge \mathsf{lab}_{\phi,\psi}$.

Proof. Most clauses follow directly from Lemma 3, clauses 1, 2, 3 and 4 use in addition Lemma 2. We prove only the most nontrivial ones.

- The "if" direction of 7. We reason by contraposition. Assume for some model $[\![A]\!]_{\mathsf{PExpr}} \neq [\![B]\!]_{\mathsf{PExpr}}$. W.l.o.g., it means there is $(x,y) \in [\![A]\!]_{\mathsf{PExpr}} - [\![B]\!]_{\mathsf{PExpr}}$. As multi-label models do not impose any restrictions on labeling, we can set $[\![v]\!]_{\mathsf{NExpr}} := \{y\}$. It will not change $[\![A]\!]_{\mathsf{PExpr}}$ and $[\![B]\!]_{\mathsf{PExpr}}$ by assumption on v. Clearly, $x \in [\![\langle A[v]\rangle]\!]_{\mathsf{NExpr}} - [\![\langle B[v]\rangle]\!]_{\mathsf{NExpr}}$, hence these two node expressions are not equivalent.

- Clause 12. The "if" direction: by Lemma 9 and Corollary 6. The converse direction: assume for some multilabel model $[\![A[\mathsf{lab}_{A,B}]]\!]_{\mathsf{PExpr}} \not\subseteq [\![B[\mathsf{lab}_{A,B}]]\!]_{\mathsf{PExpr}}$. It means there is a pair $(x,y) \in [\![A[\mathsf{lab}_{A,B}]]\!]_{\mathsf{PExpr}} - [\![B[\mathsf{lab}_{A,B}]]\!]_{\mathsf{PExpr}}$ and thus $[\![\mathsf{lab}_{A,B}]\!]_{\mathsf{NExpr}} \neq \emptyset$. By Lemma 5, this implies that $[\![A[\mathsf{lab}_{A,B}]]\!]_{\mathsf{PExpr}} = [\![A]\!]_{\mathsf{PExpr}}$ and denotations of all labels occurring in A and B are disjoint. As changing denotations of labels outside of $\Sigma(A,B)$ does not affect $[\![A]\!]_{\mathsf{PExpr}}$ and $[\![B]\!]_{\mathsf{PExpr}}$, we can use one choose one of them to label all nodes which are not label by labels in $\Sigma(A,B)$, send all the remaining ones to $\emptyset$ and thus obtain a XML countermodel for $\vDash_{\mathsf{xml}} A \equiv B$.

□

Remark 8. *Observe that clause 7 would not work for XML models because of restrictions on labeling. Take for example* $A := .[v]$, $B := .[v']$. *Even for multi-label models we can run into problems if* Σ *is assumed finite and all labels from* Σ *occur somewhere in* A, B. *This is why we assume in this paper* Σ *is infinite, i.e., we never run out of fresh labels.*

Having clarified the relationship between all the semantic notions of equivalence, we can set out to axiomatize them. Before we propose specific axioms, let us explain the general notion of axiomatization we use.

2.2 Proof Systems

We will propose proof systems for node expressions and path expressions. All equivalences will be prefixed with either $\vdash_{\mathsf{ml}}$ or $\vdash_{\mathsf{xml}}$, determining whether the equivalence in question is supposed to be a multi-label model equivalence or a XML-model equivalence, respectively.

Proof systems consist of *axioms*, *axiom schemes* and *rules*. An axiom is an identity between two path or node expressions. Examples are $\vdash_{\mathsf{ml}} \mathsf{a}^+/\mathsf{a} \cup \mathsf{a} \equiv \mathsf{a}^+$ and $\vdash_{\mathsf{ml}} \langle . [v] \rangle \equiv v$. An axiom scheme is an axiom which may contain variables ranging over path and node expressions. Thus they are patterns representing infinitely many axioms. Examples are $\vdash_{\mathsf{ml}} A/(B/C) \equiv (A/B)/C$ and $\vdash_{\mathsf{ml}} A[\phi \vee \psi] \equiv A[\phi] \cup A[\psi]$. We usually make no distinctions between axioms and axiom schemes, and call both just axioms. An inference rule is an implication between a conjunction of identities and another identity, typically containing again variables ranging over path and node expressions. An example is the transitivity rule from equational logic defined below. Sometimes such a rule has side-conditions restricting its application. We call such rules *unorthodox*. An example is

$$\text{from } \vdash_{\mathsf{ml}} \langle A[v] \rangle \equiv \langle B[v] \rangle \text{ infer } \vdash_{\mathsf{ml}} A \equiv B, \text{ provided that } v \notin \Sigma(A, B).$$

A proof based on a given proof system Γ is a finite sequence of equivalences such that each equivalence is either an axiom of Γ, or the universal tautology $\vdash_a P \equiv P$ or is obtained from preceding equivalences using the rules of Γ or the inference rules from equational logic:

- from $\vdash_a P \equiv Q$ infer $\vdash_a Q \equiv P$, (symmetry)
- from $\vdash_a P \equiv Q$ and $\vdash_a Q \equiv R$ infer $\vdash_a P \equiv R$, (transitivity)

where P, Q, R can be arbitrary node or path expressions and $\vdash_a$ is an arbitrary prefix. Finally we have the replacement rule. Because our syntax makes a difference between path and node expressions, the replacement rule has to be typed and we get two rules:.

$$\begin{array}{ll} \text{from } \vdash_{\mathsf{ml}} \phi \equiv \psi \text{ and } \vdash_{\mathsf{ml}} A(\phi) \equiv B(\phi) & \text{infer } \vdash_{\mathsf{ml}} A(\psi) \equiv B(\psi) \\ \text{from } \vdash_{\mathsf{ml}} A \equiv B \text{ and } \vdash_{\mathsf{ml}} \phi(A) \equiv \psi(A) & \text{infer } \vdash_{\mathsf{ml}} \phi(B) \equiv \psi(B). \end{array}$$

3 The axioms

This section discusses the axioms and the extra rules of our axiomatization. The axioms are presented in Table 2 and divided into four parts. As stated above, we are using prefixes $\vdash_{\mathsf{ml}}$ and $\vdash_{\mathsf{xml}}$ to denote, respectively, provable equivalences on multi-label and uni-label trees. Now we can formulate these notions precisely:

- $\vdash_{\mathsf{ml}} P \equiv Q$: $P \equiv Q$ can be proved from the equations in Table 2 and the axioms and rules of equational logic.
- $\vdash_{\mathsf{xml}} P \equiv Q$: if in addition to those, also the rules in Table 3 may be used.

In the following subsections, we describe the axioms and the rules in more detail, and we will discuss some derivable equivalences. In Section 4, we will prove the completeness of our axiomatization.

3.1 Idempotent Semirings

These axiom schemes govern the behavior of the two binary path operators /, $\cup$, the path constant "$\cdot$" and their interplay. *Idempotent semirings* is the name traditionally used in algebra. Idempotency is the axiom ISAx3. Distributive lattices are natural examples of idempotent semirings if $\cap$ is denoted as / and the identuty constant . as the boolean $\top$. Natural numbers with addition and multiplication form a semiring, but not an idempotent one, as 2+2 is not equal to 2. From our point of view, the most important example of idempotent semirings (with some additional operations) are *Tarski's relation algebras* [14, 15]. Of course, it is not surprising that our operations / and $\cup$ satisfy the laws of relation algebras: the interpretation of these operations is simply the same. For the same reason, *Kleene algebras* [8, 9] are also idempotent semirings (with additional operation $*$).

Some authors demand that semirings have an additional constant $\bot$ satisfying Der6 and Der7. For us, $\bot$ is a defined abbreviation, this is why those equivalences are derived.

3.2 Predicate Axioms

In the one-sorted signature of Tarski's relation algebras [14, 15], predicates can be treated as defined operations. This fact was used in [16] for the axiomatization of Core XPath 2.0. In Core XPath 1.0, there are less operations on relations available and predicates cannot be term-defined.PrAx1 establishes the connection between predicates, negation on node formulas and test operators $\langle\cdot\rangle$: whenever you reach a node where $\langle B\rangle$ does not succeed, you cannot proceed with B. PrAx2, PrAx4 and PrAx3 establish the interaction between predicates and remaining node and path operations: /, $\cup$, $\vee$ and the constant ".".

As a side-remark we note that instead of using predicate expressions and two-sorted signature with node and path formulas, we could use the *dynamic negation* $\sim$ introduced by Groenendijk et al. [6] and studied in [17, 7]. The axioms for *dynamic relation algebras (DRA's)* [7] with operations $\cup$, / and $\sim$ are almost equivalent to the axioms in the two groups discussed so far (idempotent semirings and predicates); the only axiom we have not introduced yet is the one saying that expressions preceded by $\sim$ are boolean. In our setting, this is ensured by axiom NdAx1 discussed below. The theoretical importance of DRA operations $\cup$, / and $\sim$ is that the operations on programs defined by means of these operations are exactly the first order definable operations that are *safe for bisimulations* (see [17] and also [2, Theorem 2.83]).

Table 2: Axioms and Axiom Schemes for $\vdash_{\mathsf{ml}}$-Equivalences

Path Axiom Schemes for Idempotent Semirings

ISAx1	$(A \cup B) \cup C$	$\equiv$	$A \cup (B \cup C)$	
ISAx2	$A \cup B$	$\equiv$	$B \cup A$	
ISAx3	$A \cup A$	$\equiv$	A	
ISAx4	$A/(B/C)$	$\equiv$	$(A/B)/C$	
ISAx5	$./A$	$\equiv$	A	
	$A/.$	$\equiv$	A	
ISAx6	$A/(B \cup C)$	$\equiv$	$A/B \cup A/C$	
	$(A \cup B)/C$	$\equiv$	$A/C \cup B/C$	

Path Axiom Schemes for Predicates

PrAx1	$A[\neg\langle B\rangle]/B$	$\equiv$	$\bot$	
PrAx2	$(A/B)[\phi]$	$\equiv$	$A/B[\phi]$	
PrAx3	$.[\langle .\rangle]$	$\equiv$	$.$	
PrAx4	$A[\phi \vee \psi]$	$\equiv$	$A[\phi] \cup A[\psi]$	

Axiom Schemes for Trees

TreeAx1	$\mathsf{a}^+/\mathsf{a} \cup \mathsf{a}$	$\equiv$	a^+	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
	$\mathsf{a}/\mathsf{a}^+ \cup \mathsf{a}$	$\equiv$	a^+	
TreeAx2	$\mathsf{a}[\phi]/\mathsf{a}^{-1}$	$\equiv$	$.[\langle \mathsf{a}[\phi]\rangle]$	for $\mathsf{a} \in \{\leftarrow, \rightarrow, \downarrow\}$
TreeAx3	$\uparrow[\phi]/\downarrow$	$\equiv$	$(\leftarrow^+ \cup \rightarrow^+ \cup .)[\langle \uparrow[\phi]\rangle]$	
TreeAx4	$(\leftarrow \cup \rightarrow)[\mathsf{root}]$	$\equiv$	$\bot$	

Node Axiom Schemes

NdAx1	ϕ	$\equiv$	$\neg((\phi \vee \psi) \vee \neg(\phi \vee \neg\psi))$	
NdAx2	$\langle .[\phi]\rangle$	$\equiv$	ϕ	
NdAx3	$\langle A \cup B\rangle$	$\equiv$	$\langle A\rangle \vee \langle B\rangle$	
NdAx4	$\langle A/B\rangle$	$\equiv$	$\langle A[\langle B\rangle]\rangle$	
NdAx5	$\langle \mathsf{a}^+[\phi]\rangle$	$\equiv$	$\langle \mathsf{a}^+[\phi \wedge \neg\langle \mathsf{a}^+[\phi]\rangle]\rangle$	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$

Table 3: Rules for Inferring $\vdash_{\mathsf{xml}}$-equivalences from $\vdash_{\mathsf{ml}}$-Node Equivalences

from	infer
$\vdash_{\mathsf{ml}} \phi \wedge \mathsf{lab}_{\phi,\psi} \equiv \psi \wedge \mathsf{lab}_{\phi,\psi}$	$\vdash_{\mathsf{xml}} \phi \equiv \psi$
$\vdash_{\mathsf{ml}} \langle A^{-1}[\mathsf{root}]\rangle \wedge \mathsf{lab}_{A,B} \equiv \langle B^{-1}[\mathsf{root}]\rangle \wedge \mathsf{lab}_{A,B}$	$\vdash_{\mathsf{xml}} A =^r B$
$\vdash_{\mathsf{ml}} \langle A[v \wedge \mathsf{lab}_{A,B}]\rangle \equiv \langle B[v \wedge \mathsf{lab}_{A,B}]\rangle$	$\vdash_{\mathsf{xml}} A \equiv B$ for a $v \notin \Sigma(A, B)$

Table 4: Derivable Equivalences for $\vdash_{\mathsf{ml}}$

Boolean Law Schemes

Der1	$\phi \vee \psi$	$\equiv$	$\psi \vee \phi$
Der2	$\phi \vee (\psi \vee \chi)$	$\equiv$	$(\phi \vee \psi) \vee \chi$

Path Equivalence Schemes

Der3	$A[\mathsf{true}]$	$\equiv$	A
Der4	$A[\mathsf{false}]$	$\equiv$	$\bot$
Der5	$(A \cup B)[\phi]$	$\equiv$	$A[\phi] \cup B[\phi]$
Der6	$A \cup \bot$	$\equiv$	A
	$\bot \cup A$	$\equiv$	A
Der7	$A/\bot$	$\equiv$	$\bot$
	$\bot/A$	$\equiv$	$\bot$
Der8	$A[\phi][\psi]$	$\equiv$	$A[\phi \wedge \psi]$

Some Other Derived Node Equivalence Schemes

Der9	$\langle A[\mathsf{false}]\rangle$	$\equiv$	false
Der10	$\langle A[\phi \vee \psi]\rangle$	$\equiv$	$\langle A[\phi]\rangle \vee \langle A[\psi]\rangle$
Der11	$\langle A[\phi \wedge \psi]\rangle \wedge \neg\langle A[\psi]\rangle$	$\equiv$	false
Der12	$\langle A[\phi]\rangle \wedge \neg\langle A[\psi]\rangle$	$\leq$	$\langle A[\phi \wedge \neg\psi]\rangle$
Der13	$\langle \mathsf{a}[\phi]\rangle$	$\equiv$	$\langle \mathsf{a}^+[\phi \wedge \neg\langle \mathsf{a}[\phi]\rangle]\rangle$ for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
Der14	$\langle A/B\rangle$	$\leq$	$\langle A\rangle$

Table 5: Equivalences of Blackburn, Meyer-Viol, de Rijke [3]

BMR0	(boolean axioms)			
BMR1	(an instance of Der9 for $A \in \mathsf{Axis}$)			
	(an instance of Der10 for $A \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$)			
BMR2	$\langle \mathsf{a}[\neg\langle \mathsf{a}^{-1}[\phi]\rangle]\rangle$	$\leq$	$\neg\phi$	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
BMR3	$\langle \mathsf{a}[\neg\phi]\rangle \wedge \langle \mathsf{a}[\phi]\rangle$	$\equiv$	false	for $\mathsf{a} \in \{\uparrow, \leftarrow, \rightarrow\}$
BMR4	$\langle \mathsf{a}[\phi]\rangle \vee \langle \mathsf{a}[\langle \mathsf{a}^+[\phi]\rangle]\rangle$	$\equiv$	$\langle \mathsf{a}^+[\phi]\rangle$	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
BMR5	$\neg\langle \mathsf{a}[\phi]\rangle \wedge \langle \mathsf{a}^+[\phi]\rangle$	$\leq$	$\langle \mathsf{a}^+[\neg\phi \wedge \langle \mathsf{a}[\phi]\rangle]\rangle$	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
BMR6	$\langle \mathsf{a}[\mathsf{true}]\rangle$	$\leq$	$\langle \mathsf{a}^+[\neg\langle \mathsf{a}[\mathsf{true}]\rangle]\rangle$	for $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
BMR7	(NdAx5 for $\downarrow$ and $\rightarrow$)			
BMR8	$\langle\downarrow[\mathsf{first} \wedge \neg\langle\rightarrow^*[\phi]\rangle]\rangle$	$\leq$	$\neg\langle\downarrow[\phi]\rangle$	
BMR9	$\langle\downarrow[\phi]\rangle$	$\leq$	$\langle\downarrow[\mathsf{first}]\rangle \wedge \langle\downarrow[\mathsf{last}]\rangle$	
BMR10	root	$\leq$	$\mathsf{first} \wedge \mathsf{last}$	

3.3 Tree Axioms

TreeAx1 is a well-known Kleene algebra axiom [8, 9]. It has little to do with the fact that we are working with trees: it simply forces a^+ to be a fixed-point containing a. However, this axiom alone would not be enough to derive that a^+ is a *smallest* fixed-point containing a. Observe that this axiom is not a scheme of infinitely many ones, as Core XPath 1.0 does not allow for transitive closure of arbitrary path expressions. TreeAx2, TreeAx3 and TreeAx4 force that $\rightarrow$ is the converse of $\leftarrow$, $\uparrow$ is the converse of $\downarrow$ and that $\uparrow$, $\leftarrow$ and $\rightarrow$ are partial functions. TreeAx3 together with TreeAx4 in addition ensure proper interplay between horizontal and vertical axes.

3.4 Node Axioms

The reader may feel suspicious about the fact that the only axiom we add to ensure that node expressions are boolean is NdAx1. This is explained in the proof of Corollary 10 below. The axioms NdAx4, NdAx2 and NdAx3 are counterparts of PrAx2, PrAx3 and PrAx4, respectively. This slight redundancy is the price we have to pay for working in a two-sorted signature. PrAx3 together with NdAx2, in addition, allow to derive node equivalences from path equivalences. NdAx5 is a very important equivalence. It is related to the fact that a^+ is not only transitive and irreflexive, but also *well-founded*: there are no infinite ascending a^+-chains. The validity of this axiom, known by logicians as *The Löb Axiom* means that a^+ allows for reasoning by induction. NdAx5 is valid for $\downarrow$ because every path in the trees we consider is of finite length and for horizontal axes because every node has finitely many children. In other words, in infinite trees NdAx5 would only be valid for $\uparrow$ (and perhaps $\leftarrow$, depending on the convention on sibling order). For more on this axiom, see [2, 18]. The latter reference explains also how to derive BMR5 from this axiom, a fact we used in our axiomatization.

3.5 Rules

The rules in Table 3 may seem counterintuitive at first sight. Their justification is provided by Lemma 7, but we admit they do feel somewhat *ad hoc*. A rule based on the clause 7 of Lemma 7 is closely related to *separability rule* in *dynamic algebras* corresponding to *logic of programs (PDL)*, see [12]. In the field of XML query languages, a rule inspired by a reasoning analogous to the one behind 8 and 7 appeared in Miklau et al. [11, Proposition 1] An alternative way of using $\vdash_{\mathsf{ml}}$-node equivalences would be to do in two steps: derive first $\vdash_{\mathsf{ml}}$-path equivalences and then derive $\vdash_{\mathsf{xml}}$-path equivalences from $\vdash_{\mathsf{ml}}$-path equivalences. Table 6 displays the alternative rules we could pose.

3.6 Derived Equivalences

Lemma 9. *All the equivalences in Tables 4 and 5 are derivable from the axioms in Table 2 and the axiom and rules of equational logic.*

We give two example derivations. We use the axiom and the rules of equational logic and the many-sorted replacement rule without explicit mention.

First we derive Der5:

$$\begin{aligned}
(A \cup B)[\phi] &\equiv ((A \cup B)/.)[\phi] && \text{by ISAx5}\\
&\equiv (A \cup B)/.[\phi] && \text{by PrAx2}\\
&\equiv A/.[\phi] \cup B/.[\phi] && \text{by PrAx4}\\
&\equiv (A/.)[\phi] \cup (B/.)[\phi] && \text{by PrAx2}\\
&\equiv A[\phi] \cup B[\phi] && \text{by ISAx5}
\end{aligned}$$

Now we can derive BMR4:

$$\begin{aligned}
\langle \mathsf{a}^+[\phi]\rangle &\equiv \langle (\mathsf{a} \cup \mathsf{a}/\mathsf{a}^+)[\phi]\rangle && \text{by TreeAx1}\\
&\equiv \langle \mathsf{a}[\phi] \cup (\mathsf{a}/\mathsf{a}^+)[\phi]\rangle && \text{by Der5}\\
&\equiv \langle \mathsf{a}[\phi] \cup \mathsf{a}/\mathsf{a}^+[\phi]\rangle && \text{by PrAx2}\\
&\equiv \langle \mathsf{a}[\phi]\rangle \vee \langle \mathsf{a}/\mathsf{a}^+[\phi]\rangle && \text{by NdAx3}\\
&\equiv \langle \mathsf{a}[\phi]\rangle \vee \langle \mathsf{a}[\langle \mathsf{a}^+[\phi]\rangle]\rangle && \text{by NdAx4}
\end{aligned}$$

An important consequence of Lemma 9 is:

Corollary 10. *All substitutions of classical propositional logic (boolean algebra) validities are derivable node equivalences.*

Proof. This follows from a recent solution to the so-called Robbins problem concerning the axiomatization of Boolean algebras [10]: any algebra where NdAx1, Der1 and Der2 hold satisfies all the boolean axioms. □

However, let us add that the task of deriving all boolean validities from NdAx1, Der1 and Der2 is highly nontrivial. It was an open problem in universal algebra for almost seventy years. The positive solution was finally obtained in the 1990's by the EQP theorem prover (Argonne National Laboratory, USA) and even today is considered one of the peak achievements in automated reasoning. It is much easier, though, to derive all the boolean axioms if NdAx1 is replaced by a very similar *Huntington equation* (see [10] and the references therein):

$$\phi \equiv \neg(\neg\phi \vee \psi) \vee \neg(\neg\phi \vee \neg\psi).$$

We finish this section with one more example. We will show $\vdash_{\mathsf{xml}} v \wedge v' \equiv \mathsf{false}$ for arbitrary pair of distinct $v, v' \in \Sigma$. Of course, this is not a valid $\vdash_{\mathsf{ml}}$-equivalence. In fact, this is the main equivalence which allows us to tell apart $\vdash_{\mathsf{ml}}$ and $\vdash_{\mathsf{xml}}$ and we considered for some time posing it as an $\vdash_{\mathsf{xml}}$-axiom. This is why we want to show it is derivable. First, observe that

$$\mathsf{lab}_{v \wedge v', \mathsf{false}} = \mathsf{A}(\neg v \vee \neg v').$$

So, to infer the equivalence in question using Table 3, we need to show that

$$\vdash_{\mathsf{ml}} v \wedge v' \wedge \mathsf{A}(\neg v \vee \neg v') \equiv \mathsf{false} \wedge \mathsf{A}(\neg v \vee \neg v').$$

$$\begin{aligned}
v \wedge v' \wedge \mathsf{A}(\neg v \vee \neg v') &\equiv v \wedge v' \wedge \neg\langle\uparrow^*[\langle\downarrow^*[v \wedge v']\rangle]\rangle && \text{by definition of } \mathsf{A} \\
&\equiv v \wedge v' \wedge \neg(v \wedge v) \wedge \neg\langle\uparrow^+[\langle\downarrow^*[v \wedge v']\rangle]\rangle && \text{by definition of } \uparrow^* \\
&\equiv \mathsf{false} && \text{by boolean laws} \\
&\equiv \mathsf{false} \wedge \mathsf{A}(\neg v \vee \neg v') && \text{by boolean laws.}
\end{aligned}$$

4 Completeness Results

Our driving result is the next completeness theorem for node-expressions whose derivation system only uses the axioms in Table 2 and the axiom and rules of equational logic. As a corrolary we obtain completeness for root and arbitrary equivalence of path expressions with respect to the derivation systems having the extra rules.

Theorem 11 (Completeness)**.** *For every valid equivalence $\vDash_{\mathsf{ml}} \phi \equiv \psi$ between node expressions, $\vdash_{\mathsf{ml}} \phi \equiv \psi$ can be derived from the axioms in Table 2 and the axiom and rules of equational logic. In fact, $\vDash_{\mathsf{ml}} \phi \equiv \psi$ iff $\vdash_{\mathsf{ml}} \phi \equiv \psi$.*

Corollary 12. *The axioms presented in Tables 2 together with rules presented in Table 3 are complete for node equivalence, root equivalence and strong path equivalence over XML models. That is:*

- $\vDash_{\mathsf{xml}} \phi \equiv \psi$ *iff* $\vdash_{\mathsf{xml}} \phi \equiv \psi$
- $\vDash_{\mathsf{xml}} A \equiv^r B$ *iff* $\vdash_{\mathsf{xml}} A \equiv^r B$
- $\vDash_{\mathsf{xml}} A \equiv B$ *iff* $\vdash_{\mathsf{xml}} A \equiv B$

Proof. Follows from Theorem 11 and Lemma 7. □

The proof of Theorem 11 consists of two steps. First we show that each node expression is provably equivalent to a simple node expression, to be defined shortly. These expressions are isomorphic variants of the modal logical tree formulas used in [3]. For instance, our $\langle\downarrow[p]\rangle$ is just the modal formula $\langle\downarrow\rangle p$, saying "I have a p–child". [3] contains a complete derivation system for these simple node expressions. The second step of our completeness proof shows that all these derivations can also be done in our system.

The *simple node expressions* (abbreviated as siNode) are defined as:

$$\begin{aligned}
\mathsf{siAxis} &:= \downarrow \mid \leftarrow \mid \uparrow \mid \rightarrow \mid \downarrow^+ \mid \leftarrow^+ \mid \uparrow^+ \mid \rightarrow^+ \\
\mathsf{siPath} &:= \mathsf{siAxis}[\mathsf{siNode}] \\
\mathsf{siNode} &:= \mathsf{true} \mid \mathsf{false} \mid v \mid \langle\mathsf{siPath}\rangle \mid \neg\mathsf{siNode} \mid \mathsf{siNode} \vee \mathsf{siNode}
\end{aligned}$$

where $v \in \Sigma$.

Lemma 13. *Every node expression ϕ is provably equivalent to a simple node expression ϕ^{s}.*

Proof. We provide a translation $(\cdot)^{\mathsf{s}} : \mathsf{NodeExpr} \mapsto \mathsf{siNode}$ which is constant for elements of siNode. This mapping uses an auxiliary mapping $(\cdot)^{\mathsf{s}} : \mathsf{PathExpr} \mapsto (\mathsf{NodeExpr} \mapsto \mathsf{NodeExpr})$ assigning to every path expression a unary function defined on node expressions s.t. for every $A \in \mathsf{PathExpr}$ and every $\phi \in \mathsf{siNode}$, $A^{\mathsf{s}}(\phi) \in \mathsf{siNode}$. As domains of both mappings are disjoint, we use the same symbol with no risk of confusion.

$$
\begin{aligned}
v^{\mathsf{s}} &:= v && \text{for } v \in \Sigma \\
(\neg\phi)^{\mathsf{s}} &:= \neg\phi^{\mathsf{s}} \\
(\phi \vee \psi)^{\mathsf{s}} &:= \phi^{\mathsf{s}} \vee \psi^{\mathsf{s}} \\
\langle A \rangle^{\mathsf{s}} &:= A^{\mathsf{s}}(\mathsf{true}) \\
.^{\mathsf{s}}(\phi) &:= \phi \\
\mathsf{a}^{\mathsf{s}}(\phi) &:= \langle \mathsf{a}[\phi] \rangle && \text{for } \mathsf{a} \in \mathsf{siAxis} \\
(A \cup B)^{\mathsf{s}}(\phi) &:= A^{\mathsf{s}}(\phi) \vee B^{\mathsf{s}}(\phi) \\
(A[\psi])^{\mathsf{s}}(\phi) &:= A^{\mathsf{s}}(\psi^{\mathsf{s}} \wedge \phi) \\
(A/B)^{\mathsf{s}}(\phi) &:= A^{\mathsf{s}}(B^{\mathsf{s}}(\phi))
\end{aligned}
$$

Hence, the Lemma can be reformulated as follows:

> For every $A \in \mathsf{PathExpr}$, $\vdash_{\mathsf{ml}} \langle A \rangle \equiv A^{\mathsf{s}}(\mathsf{true})$ and for every $\phi \in \mathsf{NodeExpr}$, $\vdash_{\mathsf{ml}} \phi \equiv \phi^{\mathsf{s}}$.

By Der3, this in turn is implied by the following

> For every $A \in \mathsf{PathExpr}$ and every $\phi \in \mathsf{NodeExpr}$, $\vdash_{\mathsf{ml}} \langle A[\phi] \rangle \equiv A^{\mathsf{s}}(\phi)$ and $\vdash_{\mathsf{ml}} \phi \equiv \phi^{\mathsf{s}}$.

Inductive steps for node expressions are obvious, hence we focus only on inductive steps for path expressions.

- $A = .$: by NdAx2.
- $A = \mathsf{a} \in \mathsf{siAxis}$: by definition of $(\cdot)^{\mathsf{s}}$.
- $A = B \cup C$:

$$
\begin{aligned}
\langle B \cup C[\phi] \rangle &\equiv \langle B[\phi] \cup C[\phi] \rangle && \text{by Der5} \\
&\equiv \langle B[\phi] \rangle \vee \langle C[\phi] \rangle && \text{by NdAx3} \\
&\equiv B^{\mathsf{s}}(\phi) \vee C^{\mathsf{s}}(\phi) && \text{by IH} \\
&\equiv (B \cup C)^{\mathsf{s}}(\phi) && \text{by definition of } (\cdot)^{\mathsf{s}}
\end{aligned}
$$

- $A = B/C$:

$$\begin{aligned}
\langle (B/C)[\phi] \rangle &\equiv \langle B/C[\phi] \rangle && \text{by PrAx2} \\
&\equiv \langle B[\langle C[\phi] \rangle] \rangle && \text{by NdAx4} \\
&\equiv B^{\mathsf{s}}(C^{\mathsf{s}}(\phi)) && \text{by IH} \\
&\equiv (B/C)^{\mathsf{s}}(\phi) && \text{by definition of } (\cdot)^{\mathsf{s}}
\end{aligned}$$

- $A = B[\psi]$:

$$\begin{aligned}
\langle B[\psi][\phi] \rangle &\equiv \langle B[\psi^{\mathsf{s}}][\phi] \rangle && \text{by IH on } \mathsf{NodeExpr} \\
&\equiv \langle B[\psi^{\mathsf{s}} \wedge \phi] \rangle && \text{by Der8} \\
&\equiv B^{\mathsf{s}}(\psi^{\mathsf{s}} \wedge \phi) && \text{by IH on } \mathsf{PathExpr} \\
&\equiv (B[\psi])^{\mathsf{s}}(\phi) && \text{by definition of } (\cdot)^{\mathsf{s}}
\end{aligned}$$

□

To make our notation more compact, we will cheat a little. As by Der3, $\mathsf{a} \equiv \mathsf{a}[\mathsf{true}]$ for every $\mathsf{a} \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$, we will use both equivalently in simple node expressions. This allows us to treat first, last, root and leaf as elements of siNode. Thus, BMR1–BMR10 can be treated as statements about simple node expressions.

Lemma 14. *The axiomatization in Table 2 is complete for simple node expressions over generalized tree models. That is, for every ϕ, $\psi \in \mathsf{siNode}$, $\vDash_{\mathsf{ml}} \phi \equiv \psi$ iff $\vdash_{\mathsf{ml}} \phi \equiv \psi$.*

Proof. This follows from Lemma 9 and the result of Blackburn et al. [3]. Modulo the difference between an equational derivation system and a modal Hilbert style axiomatization of valid formulas, they prove that the axiomatization consisting of BMR1–BMR10 is complete for simple node expressions over generalized tree models. See Theorems 5.25 and 5.27 in [2] for the relation between algebraic and modal logical proof systems. □

Lemmas 13 and 14 together immediately yield Theorem 11. Observe it was enough to prove the first claim, as $\vdash_{\mathsf{ml}} \phi \equiv \psi$ implies $\vDash_{\mathsf{ml}} \phi \equiv \psi$ can be established by a straightforward verification.

Remark 15. *The rules in Table 3 could have been replaced by Table 6.*

Table 6: Alternative Rules for Other Notions of Equivalence

from	infer	
$\vdash_{\mathsf{ml}} \phi \wedge \mathsf{lab}_{\phi,\psi} \equiv \phi \wedge \mathsf{lab}_{\phi,\psi}$	$\vdash_{\mathsf{xml}} \phi \equiv \psi$	
$\vdash_{\mathsf{ml}} \langle A^{-1}[\mathsf{root}]\rangle \equiv \langle B^{-1}[\mathsf{root}]\rangle$	$\vdash_{\mathsf{ml}} A \equiv^r B$	
$\vdash_{\mathsf{ml}} A[\langle \mathsf{lab}_{A,B}\rangle] \equiv^r B[\langle \mathsf{lab}_{A,B}\rangle]$	$\vdash_{\mathsf{xml}} A \equiv^r B$	
$\vdash_{\mathsf{ml}} \langle A[v]\rangle \equiv \langle B[v]\rangle$	$\vdash_{\mathsf{ml}} A \equiv B$	for $v \notin \Sigma(A, B)$
$\vdash_{\mathsf{ml}} A[\mathsf{lab}_{A,B}] \equiv B[\mathsf{lab}_{A,B}]$	$\vdash_{\mathsf{xml}} A \equiv B$	
$\vdash_{\mathsf{ml}} \phi \equiv \psi$	$\vdash_{\mathsf{xml}} \phi \equiv \psi$	
$\vdash_{\mathsf{ml}} A \equiv^r B$	$\vdash_{\mathsf{xml}} A \equiv^r B$	
$\vdash_{\mathsf{ml}} A \equiv B$	$\vdash_{\mathsf{xml}} A \equiv B$	

5 Conclusions

We promised a small gem, a surpise, and an open problem. The gem is of course the simple set of axioms which are complete for node expressions. The surprise is that it seems so hard to do something similar for path expressions without using a non-orthodox rule, which we leave as an open problem. Non-orthodox rules are shunned by algebraists (see e.g., Section 3 in [13] for a discussion). Our proof in its crucial part is not concerned with path expressions—we are working with node expressions instead and then use non-orthodox rules to derive path equivalences. It would be more satisfying to replace the "node equivalence engine" at the heart of our proof by genuinely path-oriented relational reasoning. Unfortunately, this proved more difficult than we expected.

Still, we have a complete axiomatization. A completeness result like this can be considered a meta-result: if a prover or query optimization algorithm can handle all the axioms and recognizes the validity of all inference rules in our axiomatization, it can handle all valid XPath equivalences.

Acknowledgments

The first and the second author wish to gratefully acknowledge the support of, respectively, Veni and Rubicon grants of the Netherlands Organization for Scientific Research (NWO), grant numbers 639.021.508 and 680-50-0613.

References

[1] M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, 2005.

[2] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[3] P. Blackburn, W. Meyer-Viol, and M. de Rijke. A proof system for finite trees. In H. Kleine Büning, editor, *Computer Science Logic*, volume 1092 of *LNCS*, pages 86–105. Springer, 1996.

[4] G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. LICS*, Copenhagen, 2002.

[5] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS'03*, pages 179–190, 2003.

[6] Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14(1):39–100, 1991.

[7] Marco Hollenberg. An equational axiomatization of dynamic negation and relational composition. *Journal of Logic, Language and Information*, 6(4):381–401, 1997.

[8] S. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.

[9] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.

[10] W. McCune. Solution of the Robbins problem. *J. Autom. Reason.*, 19(3):263–276, 1997.

[11] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. PODS'02*, pages 65–76, 2002.

[12] V. Pratt. Dynamic algebras: Examples, constructions, applications. *Studia Logica*, 50, 3–4:571–605, 1991.

[13] Mark Reynolds. An axiomatization for Until and Since over the reals without the IRR rule. *Studia Logica*, 51(2):165–193, 1992.

[14] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[15] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41. AMS Colloquium publications, Providence, Rhode Island, 1987.

[16] B. ten Cate and M. Marx. Axiomatizing the logical core of XPath 2.0. In Th. Schwentick and D. Suciu, editors, *Proceedings ICDT 2007*, 2007.

[17] Johan van Benthem. Program constructions that are safe for bisimulation. *Studia Logica*, 60(2):311–330, 1998.

[18] Johan van Benthem. Modal frame correspondence and fixed-points. *Studia Logica*, 83(1):133–155, 2006.

Teaching SQL at USQ

Stijn Dekeyser
University of Southern Queensland, Australia

Abstract

In a *Liber Amicorum* for Jan Paredaens, a contribution about teaching an introductory course in database systems seems almost indispensable. We briefly describe some differences between teaching in Belgium and Australia, and present a tool developed in-house to help teach SQL, dubbed *SQLify*.

Introduction

Teaching programming and query languages is one of Jan's enduring passions. His approach to teaching SQL starts by introducing relational algebra and calculus and then focusses on a series of ever-more intricate query examples. Just as Date introduced the now-famous *Supplier–Parts* database to teach the various SQL constructs [2], Jan is fond of the equally ubiquitous *Bars–Drinkers–Beers* schema (attributed to J.D. Ullman). Consisting of three binary relations each representing a many-to-many relationship between three entities, Jan's example schema lends itself to some very complicated (and exceptionally entertaining!) query problems. It is unclear to me whether students fully appreciate the intricacies Jan confronts them with when they first take the course, but I can personally attest to learning from them much later in life.

Teaching at USQ

One of the differences between Anglosaxon and some Northwestern European academic education systems is that the latter lack the notion of assignments. This is a sort of paradisiacal situation for faculty in those countries as they are not faced with endless hours of marking, or (if they are lucky enough to have assistants to do the marking) constructing assignment questions, marking criteria and feedback forms. The downside is that students typically don't get tested on their SQL skills until the day of the final exam, for which most started studying only a few days earlier. The quality of answers you usually get in such a situation is rather low, a problem exacerbated by the fact that students don't really know what to expect both in terms of questions and in terms of marking criteria.

At Anglosaxon institutions such as the *University of Southern Queensland* the application of sound education theories and techniques is seen as a major selling

point to attract students. In addition, a significant portion of its students are studying in external modus, often on other continents. For these reasons, assignments are an important ingredient in students' and faculty's lives, and also receive a lot of attention from the academic leadership. There are training sessions on creating objectives and marking criteria, establishing good marking procedures, and minimizing marking time while maximizing student outcomes (which are clearly competing goals). Any fundamental research and practical software that helps achieve any of these goals is welcomed.

Introducing *SQLify*

Inspired by my experiences at the universities of Antwerp and Southern Queensland, I set out with a colleague to devise a web-based application that helps teach SQL and also helps assessment of SQL queries in assignments. The goals that drove development of *SQLify* are:

1. Provide rich feedback on SQL statements to students in both an automated and a semi-automated fashion;
2. Use database theory effectively to partially automate assessment of queries;
3. Facilitate and manage peer-review to enhance learning outcomes for students in two ways: receiving feedback from multiple sources, and conducting reviews of other students' work;
4. Combine peer-review and automated assessment to yield a wider range of final marks (beyond binary marking);
5. Automatically judge the accuracy of reviews performed by students for other students;
6. Reduce the number of necessary interventions from instructors, freeing them for other forms of teaching.

While our software is not the first to address the teaching of SQL, it is probably the most advanced in combining ideas from the scientific fields of Computing Education and Databases. Figure 1 compares *SQLify* to other systems.

SQLify was implemented by a graduate student and is being trialled for use in our Database Systems course. We have written a few conference papers focussing on various aspects of the system, and also published a comprehensive journal article at *Informatics in Education* [3]. Future development of the system will focuss on supporting Relational Algebra and simple Datalog expressions.

References

[1] Paul De Bra, Henk Olivié, and Jan Paredaens. *Leren programmeren met Pascal.* Kluwer Bedrijfswetenschappen, Deventer, 1992.

Feature	*eSQL*	*SQL-Tutor*	*SQLator*	*AsseSQL*	*SQLify*
Modelling of student to individualize instructional sessions	✗	✓	✗	✗	✗
Visualization of database schema	✗	✓	✗	✗	✓
Visualization of query processing	✓	✗	✗	✗	✓
Feedback on query semantics	✗	✓	✗	✗	✓[a]
Automatic assessment (using heuristics)	✗	✗	✓	✓[b]	✓[c]
Automatic assessment (using CQ query equivalence)	✗	✗	✗	✗	✓
Use of peer review for assessment	✗	✗	✗	✗	✓
Relational Algebra expressions support	✗	✗	✗	✗	✓[d]
Special treatment of DISTINCT and ORDER BY	✗	✗	✗	✗	✓
SQL-injection attack countermeasures	✗	✗	✗	✗	✓

Figure 1: Comparison of existing tools and *SQLify*. (a) in practice mode only. (b) on two instances (proposal only). (c) for queries not in CQ. (d) planned for next version.

[2] Chris Date. *An Introduction to Database Systems,* 8th edition. Addison-Wesley, 2004.

[3] Stijn Dekeyser, Michael de Raadt, and Tien Yu Lee. *A system employing peer review and enhanced computer assisted assessment of querying skills.* In *Informatics in Education*, **6** (1). pp. 163-178. ISSN 1648-5831, 2007.

Completeness of Database Query Languages

George H.L. Fletcher
School of Engineering and Computer Science
Washington State University
Vancouver, WA, USA
gfletcher@acm.org

Jan Paredaens' seminal 1978 results in *Information Processing Letters* on the expressive power of the relational algebra were among the first significant bridges between the model theory and database research communities. Together with those of Bancilhon in that same year for the relational calculus, Paredaens' results provided the database community with a deep new perspective on investigating the completeness, and, in general, the semantics of database query languages. This talk will give a brief historical overview of fundamental completeness results in the three decades since the *IPL* paper, spanning query languages for the relational, nested relational, and XML data models. Ongoing research on leveraging these results in data integration and XPath query processing will also be briefly highlighted.

Tree-structured object creation in database transformations

Jan Van den Bussche

Abstract

Within the class of "determinate" object-creating database transformations, identified by Abiteboul and Kanellakis, a natural proper subclass consists of the so-called "constructive" transformations. A determinate transformation is constructive if its input-output pairs satisfy a condition discovered by Jan Paredaens. In this note we point out that when object creation is "tree-structured," as is the case in tree-structured data models such as XML, determinate transformations are *always* constructive.

Dedicated to Jan Paredaens
for his 60th birthday

1 Introduction

In tree-structured data models, such as in the XML data model [15, 17], it is desirable that transformations can be expressed that are *object-creating*, meaning that the result of a transformation can contain objects (in this case, tree nodes) that do not appear in the input database. Indeed, the standard XML query language XQuery [16] allows the expression of object-creating queries by means of the element construction operation.

Well before the rise of the XML data model, however, general database transformations (possibly object-creating, and possibly non-deterministic) were already studied by Abiteboul and Vianu [3, 4], and Abiteboul and Kanellakis [2] introduced the class of "determinate" transformations as those that are non-deterministic only in the choice of the id's of the new objects. Abiteboul and Kanellakis also introduced a very natural query language, called IQL, for expressing general determinate transformations. IQL is equivalent to the relational algebra extended with three programming constructs: object creation; assignment to relation variables; and while-loops. At around the same time, another equivalent language, called GOOD, was introduced by Jan Paredaens and his collaborators [10].

Not all determinate transformations are expressible by an IQL program, however. There even exist single input-output pairs of database instances that cannot be realized by any IQL program. This situation motivated Jan Paredaens to formulate a condition on pairs (I, J) of database instances that is necessary

and sufficient for J to be an output of some GOOD (or IQL) program applied to I [6]. This condition states the existence of an extension homomorphism from the group of automorphisms of I to the group of automorphisms of J, and generalizes to object creation Jan Paredaens's earlier result on the BP-completeness (Bancilhon-Paredaens) of the relational algebra [13, 7, 9]. Later, the naturalness of the extension homomorphism condition was confirmed when it was shown that the IQL-expressible transformations are precisely those determinate transformations all of whose input-output pairs admit an extension homomorphism [14].

The developments just described happened largely before the rise of the XML data model. In this note, we ask ourselves what happens when object creation is tree structured, i.e., the newly created objects in the output form a tree, the leafs of which are labeled by objects from the input (a precise definition will be given later). XML is clearly tree-structured. We will show that in that case, the gap between determinate and IQL-expressible vanishes, i.e., the extension homomorphism condition is always satisfied for tree-structured object creation.

2 Database transformations

We recall some essential definitions in this section, largely taken from our earlier paper [14]. For background and motivation for these definitions, see Abiteboul, Hull and Vianu [1].

It is assumed that an infinite collection of *relation names* is given. To each relation name R a natural number $\alpha(R)$ is associated, called the *arity* of R, such that each number is the arity of infinitely many relation names. A *database schema* is a finite set of relation names.

It is furthermore assumed that a countably infinite universe $\mathbf{U}$ of abstract data elements, called *objects*, is given.

An *instance* I of a database schema $\mathcal{S}$ is a finite relational structure of type $\mathcal{S}$, consisting of a finite subset $|I|$ of $\mathbf{U}$, called the *domain*, and a mapping on $\mathcal{S}$, assigning to each relation name R of $\mathcal{S}$ a relation R^I on $|I|$ of rank $\alpha(R)$ (i.e., a subset of $|I|^{\alpha(R)}$), called the *content* of R. The set of all database instances of the schema $\mathcal{S}$ is denoted by $\mathrm{inst}(\mathcal{S})$.

Let $\mathcal{S}_{\mathrm{in}}$ and $\mathcal{S}_{\mathrm{out}}$ be two database schemas. A *determinate transformation from* $\mathcal{S}_{\mathrm{in}}$ *to* $\mathcal{S}_{\mathrm{out}}$ is an input-output relationship $Q \subseteq \mathrm{inst}(\mathcal{S}_{\mathrm{in}}) \times \mathrm{inst}(\mathcal{S}_{\mathrm{out}})$ satisfying the following three properties:

1. If $Q(I, J)$ then $|I| \subseteq |J|$;
2. If $Q(I, J)$ and f is a permutation of $\mathbf{U}$, then also $Q(f(I), f(J))$, where by $f(I)$ we mean the database instance obtained from I by applying f pointwise to all objects occurring in I;
3. If $Q(I, J_1)$ and $Q(I, J_2)$, then $J_2 = f(J_1)$ for some permutation f of $\mathbf{U}$ that is the identity on $|I|$.

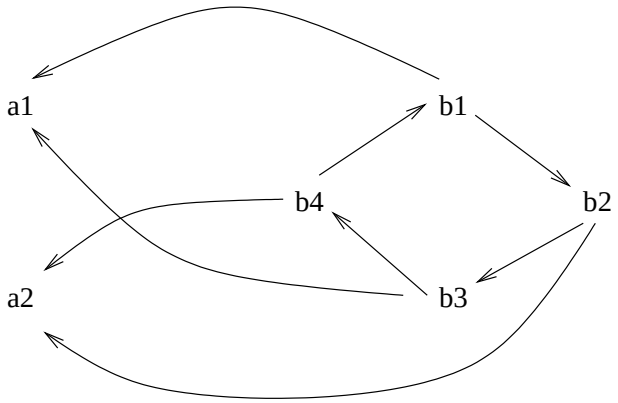


Figure 1: Instance I consists just of the two a's. Instance J adds the four b's together with the arrows (stored in a binary relation). There does not exist an extension homomorphism from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$.

The first requirement above is technical but harmless. The second one is a classical consistency criterion [9, 5] known as *genericity* [12]. The third, finally, expresses the determinacy property [2].

For a database instance I, we denote by $\mathrm{Aut}(I)$ the set of all permutations f of $|I|$ for which $f(I) = I$; such permutations are called automorphisms of I. The set $\mathrm{Aut}(I)$, with the operation of composition, forms a group structure.

A crucial concept, introduced by Jan Paredaens [6], now is the following. Let I and J be database instances such that $|I| \subseteq |J|$. An *extension homomorphism* from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$ is a group homomorphism $h : \mathrm{Aut}(I) \to \mathrm{Aut}(J)$ such that for each $f \in \mathrm{Aut}(I)$, the permutation $h(f)$ is an extension of f, i.e., $h(f)$ agrees with f on $|I|$.

We now call a determinate transformation Q *constructive* [14] if for every input-output pair (I, J) of Q, there exists an extension homomorphism from $\mathrm{Aut}(I)$ to $\mathrm{Aut}(J)$. An example (due to Serge Abiteboul [2]) of a pair of instances (I, J) that does *not* admit an extension homomorphism is shown in Figure 1. As a consequence, no transformation that contains (I, J) as an input-output pair can be constructive.

3 Tree-structured transformations

Note that the output of the non-constructive example from Figure 1 has an intrinsic cyclicity to it. That observation, and the recent interest in tree-structured data models such as XML, motivates us to study tree-structured transformations as a special class of determinate transformations. We first define this class formally and then prove that tree-structured determinate transformations are always constructive.

Definition 1 *Let J be an instance of some database schema $\mathcal{S}$, and let $T \in \mathcal{S}$ be a binary relation name. We call J* tree-structured by T *if J has the following two properties:*

1. *Let V equal the set of objects occurring in T^J, and consider this binary relation T^J as a directed graph on vertex set V. Then T^J must look like a set of rooted trees; more specifically, every vertex must have at most one incoming edge, and there must be no cycles.*

2. *For every relation name $R \in \mathcal{S}$ different from T, every tuple in the relation R^J must contain at most one occurrence of a vertex, i.e., an object from V.*

The first property in the above definition is, we hope, intuitive. The intuition behind the second property is that the tuples in the relations other than T serve as "annotations" or "labels" for the various tree vertices. We can formalize labels as follows:

Definition 2 *Let J be a database instance, tree-structured by T. Let x be a vertex of J. A* label *of x is any triple of the form $(R, i, \hat{t})$, where*

- *R is a relation name of J's database schema, with $R \neq T$;*
- *t is a tuple in R^J in which x appears;*
- *i is the position in T where x appears; and*
- *$\hat{t}$ is the subtuple of t obtained by omitting x.*

When two T-vertices have precisely the same set of labels, we call them duplicates. *We call J* duplicate-free *if there are no duplicate leafs in J, where a leaf is a vertex without outgoing edges in T^J.*

Our central notion is now the following:

Definition 3 *Let Q be a determinate transformation from $\mathcal{S}_{\text{in}}$ to S_{out}, and let T be a binary relation name in S_{out}. We call Q* tree-structured by T *if for every input-output pair (I, J) of Q, the output J is tree-structured by T, with vertex set equal to $|J| - |I|$ (i.e., the set of newly created objects), and J is also duplicate-free.*

The requirement that J be duplicate-free is mainly for technical reasons. A transformation that is not duplicate-free can be easily made so by adding additional auxiliary nodes and labels.

The purpose of this note is to point out the following:

Theorem 1 *Every tree-structured determinate transformation is constructive.*

4 HF-transformations

To prove the theorem, we recall some further definitions [14].

Let D be a subset of $\mathbf{U}$. The set $\mathrm{HF}(D)$ of *hereditarily finite sets (HF-sets) with ur-elements in D* [8] is the smallest set with the property that each finite subset of $D \cup \mathrm{HF}(D)$ is itself an element of $\mathrm{HF}(D)$.

An *HF-instance* I is defined as an ordinary instance, the only difference being that the domain $|I|$ is a subset of $\mathbf{U} \cup \mathrm{HF}(\mathbf{U})$ instead of $\mathbf{U}$. The set of all HF-instances of some database schema $\mathcal{S}$ is denoted by $\mathrm{HFinst}(\mathcal{S})$. If f is a permutation of $\mathbf{U}$ and I is a HF-instance, then $f(I)$ denotes the HF-instance obtained from I by applying f pointwise to all objects appearing in I, even if they appear within HF-sets.

For database schemas $\mathcal{S}_{\mathrm{in}}$ and $\mathcal{S}_{\mathrm{out}}$, an *HF-transformation* from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$ is a partial function $Q : \mathrm{inst}(\mathcal{S}_{\mathrm{in}}) \to \mathrm{HFinst}(\mathcal{S}_{\mathrm{out}})$ such that for each I for which $Q(I)$ is defined, we have

1. $|Q(I)| \subseteq |I| \cup \mathrm{HF}(|I|)$; and
2. for any permutation f of $\mathbf{U}$, also $Q(f(I))$ is defined, and equals $f(Q(I))$.

An ordinary instance I is said to be *isomorphic* to an HF-instance I' if there is a bijection f from $|I|$ to $|I'|$ such that $f(I) = I'$. Then a determinate transformation Q from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$ is said to be *isomorphic* to an HF-transformation Q' from $\mathcal{S}_{\mathrm{in}}$ to $\mathcal{S}_{\mathrm{out}}$, if Q' is defined precisely on all instances I for which there is an output instance J such that $Q(I, J)$, and all such J are isomorphic to $Q'(I)$.

We now recall the following connection between constructive transformations and HF-transformations:

Proposition 1 ([14]) *A determinate transformation is constructive if and only if it is isomorphic to some HF-transformation.*

Hence, in order to prove our theorem, it suffices to show that every tree-structured transformation is isomorphic to some HF-transformation. Thereto, let Q be a tree-structured transformation and let $Q(I, J)$. Note that J is tree-structured; we are going to define, for each vertex x of J, the *stamp* of x by bottom-up induction as follows. Let L be the set of labels of x. Now if x is a leaf, then the stamp of x is the ordered pair $(L, \emptyset)$. If x is not a leaf, we may assume by induction that the stamps for its children have already been defined; let C be the set of all those children's stamps. Then the stamp of x is the ordered pair (L, C).

We now observe that we can consider a stamp to be a HF-set with ur-elements in $|I|$. Indeed, ordered pairs, triples, and tuples, can be unambiguously represented by HF sets [11]. The same is true of natural numbers, so the numbers i that occur in labels can also be represented. Finally, by numbering the relation names of $\mathcal{S}_{\mathrm{out}}$, we can represent the relation names that occur in labels also by numbers. Now denote by J', the HF-instance obtained from J by replacing each vertex by its stamp, and define the HF-transformation

Q' by $Q'(I) := J'$. Since Q is determinate, Q' is well-defined, i.e., the definition of $Q'(I)$ does not depend on the chosen J. Moreover, Q' is a valid HF-transformation, by the genericity of Q. Furthermore, by definition, it is clear that Q is isomorphic to Q'. We thus have our desired HF-transformation and the theorem is proved.

5 Epilogue

This note was written mainly as an excuse to recall some of the research from the good old times when I was Jan Paredaens's PhD student and we were together interested in the foundations of object identity in query languages. I remain forever grateful to Jan for the chances he gave me, for the freedom he allowed me, for the confidence he put in me, and the patience he had with an often too self-absorbed, too ambitious, sometimes even too aggressive youngster!

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul and P.C. Kanellakis. Object identity as a query language primitive. *Journal of the ACM*, 45(5):798–842, 1998.

[3] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, 1990.

[4] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124, 1991.

[5] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *Conference Record, 6th ACM Symposium on Principles of Programming Languages*, pages 110–120, 1979.

[6] M. Andries and J. Paredaens. On instance-completeness of database query languages involving object creation. *Journal of Computer and System Sciences*, 52(2):357–373, 1996.

[7] F. Bancilhon. On the completeness of query languages for relational data bases. In *Proceedings 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, 1978.

[8] J. Barwise. *Admissible Sets and Structures*. Springer-Verlag, 1975.

[9] A. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[10] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), 1994.

[11] P. Halmos. *Naive Set Theory*. Van Nostrand Reinhold, 1960.

[12] R. Hull and C.K. Yap. The format model, a theory of database organization. *Journal of the ACM*, 31(3):518–537, 1984.

[13] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, 1978.

[14] J. Van den Bussche, D. Van Gucht, M. Andries, and M. Gyssens. On the completeness of object-creating database transformation languages. *Journal of the ACM*, 44(2):272–319, 1997.

[15] Extensible markup language (XML) 1.0 (second edition). W3C Recommendation 6 October 2000.

[16] XQuery 1.0: An XML query language. W3C Recommendation 23 January 2007.

[17] XQuery 1.0 and XPath 2.0 data model. W3C Recommendation 23 January 2007.

Towards an Axiomatization of the Relational Lattice

Jan Hidders

Abstract

There have been several attempts to replace the relational algebra with another set of operations that allow for a more effective set of algebraic identities for proving equivalence and optimizing expressions. In this paper we study a proposal that is based on the observation that the *natural join* and the *inner union* satisfy the laws for lattices. Extended with special constants for empty relations and equality relations they allow us to express the SPJRU-queries, i.e., queries that can be expressed in the relational algebra with the the selection, projection, join, renaming and union operators. We present a set of algebraic semi-equations that are conjectured to axiomatize equivalence of such expressions.

For Jan Paredaens, a teacher and a mentor who enabled me to do in life what I like doing best. – JH

1 Introduction

With the introduction of the relational model [Codd, 1970] a corresponding algebra, the Relational Algebra, was introduced for manipulating relations. This algebra, however, was not defined algebraically, i.e., in terms of algebraic identities, but rather as a set of operations over relations that happened to exhibit certain algebraic properties. These properties are important for query optimization because they allow the investigation of alternative formulations of a query that might allow a more efficient evaluation. In addition these properties can also be used to reason over data dependencies if these are formulated as equations between algebra expressions, as for example was investigated for *algebraic dependencies* [Yannakakis and Papadimitriou, 1982]. For a comprehensive overview of research on axiomatizing data dependencies the reader is referred to [Abiteboul et al., 1995].

An earlier approach that started from an algebraic perspective were the Cylindric Algebras [Henkin et al., 1971, Henkin et al., 1985] as introduced by Tarski. Fortunately, as was shown in [Imielinski and Lipski, 1982], there exist close relationships between the two types of algebra. However, this also means that some negative results about non-axiomatizability in Cylindric Algebras also transfer to the Relational Algebra, even for subsets of the algebra

[Düntsch and Mikulás, 2007]. As a consequence, and also to simplify in general reasoning over the algebra, there have been alternative proposals that are based different operators. For example [Date and Darwen, 2000] propose to base the algebra on an *AND* operator, representing the natural join, and an *OR* operator that represents the union that before taking the union extends its operands with extra columns so they have the same header and fill these columns with all values from the domain associated with those columns. A similar proposal for the generalization of the union is made in [Imielinski and Lipski, 1982]. In this paper we investigate an alternative [Tropashko, 2005, Spight and Tropashko, 2006] where the union is generalized by an *inner union* that projects the operands on the columns they have in common. As will be discussed in the next session this pair of operators has the interesting properties that it satisfies the laws of a lattice as defined in Lattice Theory.

2 The Relational Lattice

We begin with the definition of some basic terminology. We postulate an infinite countable set of attribute names $\mathcal{A}$ and an infinite countable set of domain values $\mathcal{D}$. When giving examples we will usually assume that $\mathcal{D}$ is the set of natural numbers. A *header* is a finite subset of $\mathcal{A}$. The set of all headers is denoted as $\mathcal{H}$. A *tuple over header* H is a function $x : H \to \mathcal{D}$. The set of all tuples over a header H is denoted as $\mathcal{T}_H$. The *restriction of a tuple* x on a header H is denoted as $x[H]$ and defined such that $x[H] = \{(a, v) \in x \mid a \in H\}$. We generalize this to the *projection of a set of tuples* B on a header H which is $B[H] = \{x[H] \mid x \in B\}$. Both for restriction and projection we will write $[\{a, b, c\}]$ simply as $[a, b, c]$. A *relation* is a pair $r = (H, B)$ where $H \subseteq \mathcal{H}$ is the header of the relation and $B \subseteq \mathcal{T}_H$ the body of the relation. The set of all relations is denoted as $\mathcal{R}$. Note that the header is part of the relation so the empty relation with header $\{a, b\}$, i.e., $(\{a, b\}, \emptyset)$ is not equal to the empty relation with header $\{b, c\}$, i.e., $(\{b, c\}, \emptyset)$.

In examples we will often present relations as tables. For example, the relations $r_1 = (\{a, b\}, \{\{(a, 1), (b, 2)\}, \{(a, 3), (b, 4)\}\})$, $r_2 = (\{a, b\}, \emptyset)$, $r_3 = (\emptyset, \{\emptyset\})$, $r_4 = (\emptyset, \emptyset)$ are represented as follows:

$$r_1 = \begin{array}{|cc|}\hline \mathbf{a} & \mathbf{b} \\ \hline 1 & 2 \\ 3 & 4 \\ \hline \end{array} \qquad r_2 = \begin{array}{|cc|}\hline \mathbf{a} & \mathbf{b} \\ \hline\hline \end{array} \qquad r_3 = \begin{array}{|c|}\hline \\ \hline \\ \hline \end{array} \qquad r_4 = \begin{array}{|c|}\hline \\ \hline\hline \end{array}$$

We are now ready to define the two fundamental operations of the Relational Lattice. For the relations $r = (H_r, B_r)$ and $s = (H_s, B_s)$ we define the *natural join*, denoted as $r \otimes s$, and *inner union*, denoted as $r \oplus s$, such that:

$$\begin{aligned} r \otimes s &= (H_r \cup H_s, \{x \in \mathcal{T}_{H_r \cup H_s} \mid x[H_r] \in B_r \wedge x[H_s] \in B_s\}) \\ r \oplus s &= (H_r \cap H_s, \{x \in \mathcal{T}_{H_r \cap H_s} \mid x \in B_r[H_s] \vee x \in B_s[H_r]\}) \end{aligned}$$

As an illustration we offer the following two examples:

a	b
1	1
2	2
3	2
3	3

$\otimes$

b	c
1	1
2	2
2	3
4	4

$=$

a	b	c
1	1	1
2	2	2
2	2	3
3	2	2
3	2	3

a	b
1	1
2	2
3	2
3	3

$\oplus$

b	c
1	1
2	2
2	3
4	4

$=$

b
1
2
3
4

Alternatively we can also define these operations on the basis of a partial order $\sqsubseteq$ over $\mathcal{R}$ which is defined such that $(H_r, B_r) \sqsubseteq (H_s, B_s)$ iff $H_s \subseteq H_r$ and $B_r[H_s] \subseteq B_s$. It is not hard to see that this indeed defines a partial order and that $\oplus$ and $\otimes$ define the *supremum*, i.e., the least common upper bound, and *infimum*, i.e., the greatest common lower bound, with respect to this partial order[1]. As a consequence these operations satisfy the algebraic identities for a lattice which are those that define *semi-lattices*, i.e., the operations are *idempotent*, *commutative* and *associative*:

$$r \otimes r = r \quad (1) \qquad r \oplus r = r \quad (4)$$
$$r \otimes s = s \otimes r \quad (2) \qquad r \oplus s = s \oplus r \quad (5)$$
$$r \otimes (s \otimes t) = (r \otimes s) \otimes t \quad (3) \qquad r \oplus (s \oplus t) = (r \oplus s) \oplus t \quad (6)$$

and the operations satisfy the *absorption* laws:

$$r \otimes (r \oplus s) = r \quad (7) \qquad r \oplus (r \otimes s) = r \quad (8)$$

It also follows that $r \sqsubseteq s$ iff $r \otimes s = r$ iff $r \oplus s = s$.

As is well known from Lattice Theory rules 1 and 4 are in fact redundant since $r \otimes r =^{7,8} (r \otimes (r \oplus s)) \otimes (r \oplus (r \otimes s)) =^{2,3} (r \oplus s) \otimes (r \otimes (r \oplus (r \otimes s))) =^{7} (r \oplus s) \otimes r =^{2} r \otimes (r \oplus s) =^{7} r$ and a similar argument shows that $r \oplus r = r$ can be derived using rules 5, 6, 7 and 8.

Also well known from Lattice Theory is that if they exist then the *top element*, i.e., a relation $\top$ such that for all relations r it holds that $\top \oplus r = \top$, and the *unit element for* $\otimes$, i.e., the relation $1^{\otimes}$ such that $r \otimes 1^{\otimes} = r$ for all relations r, and the *zero element for* $\oplus$, i.e., the relation $0^{\oplus}$ such that $r \oplus 0^{\oplus} = 0^{\oplus}$ for all relations r, are identical. In the Relational Lattice it exists in the form of the relation $(\emptyset, \{\emptyset\})$ which we will denote in our algebra by the constant 1. Hence the following identities hold:

$$r \otimes 1 = r \quad (9) \qquad r \oplus 1 = 1 \quad (10)$$

Note that one of the two preceding rules is redundant because one can be derived using the other by $r \otimes 1 =^{10} r \otimes (r \oplus 1) =^{7} r$ and $r \oplus 1 =^{9} (r \otimes 1) \oplus 1 =^{2,5,8} 1$.

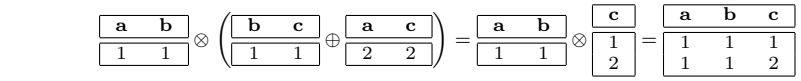
A similar relationship holds between dual of the aforementioned concepts, i.e., the *bottom element* $\bot$, the *unit element* $1^{\oplus}$ *for* $\otimes$ and the *zero element* $0^{\otimes}$ *for* $\otimes$ are identical if they exist. There is however no such element in the Relational Lattice as we defined it since it would have to be a relation with an

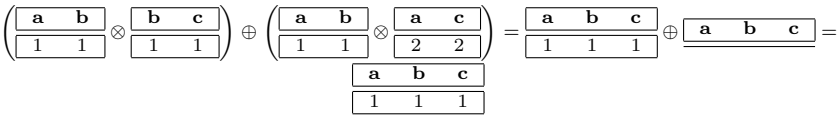
[1]The infimum and supremum are usually called the *meet* and the *join* but these terms can be somewhat confusing here since in the Relational Lattice the natural join is in fact the meet and not the join.

infinitely large header. If we would have allowed such headers then $(\mathcal{A}, \emptyset)$ would be the relation in question, however, in this paper we consider only relations with finite headers.

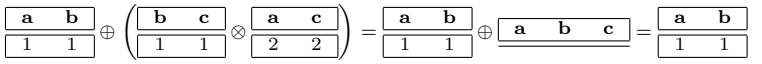
3 Distributivity

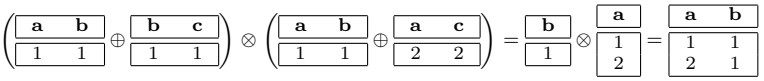
The Relational Lattice is unfortunately not a distributive lattice, i.e., it is not true in general that $r \otimes (s \oplus t) = (r \otimes s) \oplus (r \otimes t)$, as is shown by the following example





and it it also not in general true that $r \oplus (s \otimes t) = (r \otimes s) \oplus (r \otimes t)$, as is shown by the following example





However it can be shown that under certain restrictions on the headers of the involved relations distributivity holds. Consider the relations $r = (H_r, B_r)$, $s = (H_s, B_s)$ and $t = (H_t, B_t)$. We can then show show that under the assumption that $H_r \cap H_s = H_r \cap H_t$ it follows that $r \otimes (s \oplus t) = (r \otimes s) \oplus (r \otimes t)$. It is clear, even without the assumption, that the headers left and right are the same because $H_{r \otimes (s \oplus t)} = H_r \cup (H_s \cap H_t) = (H_r \cup H_s) \cap (H_r \cup H_t) = H_{(r \otimes s) \oplus (r \otimes t)}$ and we will call this header H'. If we now assume that $H_r \cap H_s = H_r \cap H_t$ then it can be shown that $B_{r \otimes (s \oplus t)} = B_{(r \otimes s) \oplus (r \otimes t)}$ as follows. To simplify the proof we pick a single attribute for each nonempty subset of $\{r, s, t\}$ that represents the attributes that appear only in the headers of the relations in the subset. For $\{r\}$ we pick a, for $\{s\}$ we pick b, for $\{t\}$ we pick c, for $\{s, t\}$ we pick d and for $\{r, s, t\}$ we pick e. Note that because of the assumption there are no variables for $\{r, s\}$ and $\{r, t\}$. Summarizing we have the headers $H_r = \{a, e\}$, $H_s = \{b, d, e\}$, $H_t = \{c, d, e\}$ and $H' = \{a, d, e\}$. Then, for every tuple $x \in \mathcal{T}_{\{a,d,e\}}$ it holds

that

$$
\begin{aligned}
& x \in B_{r\otimes(s\oplus t)} \\
\Leftrightarrow\ & x[a,e] \in B_r \wedge x[d,e] \in B_{s\oplus t} \\
\Leftrightarrow\ & x[a,e] \in B_r \wedge (x[d,e] \in B_s[d,e] \vee x[d,e] \in B_t[d,e]) \\
\Leftrightarrow\ & (x[a,e] \in B_r \wedge x[d,e] \in B_s[d,e]) \vee (x[a,e] \in B_r \wedge x[d,e] \in B_t[d,e]) \\
\Leftrightarrow\ & (x[a,e] \in B_r \wedge (\exists x' \in \mathcal{T}_{\{a,b,d,e\}} : x'[b,d,e] \in B_s \wedge x'[a,d,e] = x)) \vee \\
& (x[a,e] \in B_r \wedge (\exists x' \in \mathcal{T}_{\{a,c,d,e\}} : x'[c,d,e] \in B_t \wedge x'[a,d,e] = x)) \\
\Leftrightarrow\ & (\exists x' \in \mathcal{T}_{\{a,b,d,e\}} : x'[a,e] \in B_r \wedge x'[b,d,e] \in B_s \wedge x'[a,d,e] = x) \vee \\
& (\exists x' \in \mathcal{T}_{\{a,c,d,e\}} : x'[a,e] \in B_r \wedge x'[c,d,e] \in B_t \wedge x'[a,d,e] = x) \\
\Leftrightarrow\ & (\exists x' \in B_{r\otimes s} : x'[a,d,e] = x) \vee (\exists x' \in B_{r\otimes t} : x'[a,d,e] = x) \\
\Leftrightarrow\ & x \in B_{r\otimes s}[a,d,e] \vee x \in B_{r\otimes t}[a,d,e] \\
\Leftrightarrow\ & x \in B_{(r\otimes s)\oplus(r\otimes t)}.
\end{aligned}
$$

It is not hard to see how this proof can be generalized for arbitrary headers H_r, H_s and H_t that satisfy the assumption.

In a similar fashion it can be shown that $r\oplus(s\otimes t) = (r\oplus s)\otimes(r\oplus t)$ holds if we assume that $H_r \cap H_s = H_r \cap H_t = H_s \cap H_t$. Again it is clear that the headers are the same since $H_{r\oplus(s\otimes t)} = H_r \cap (H_s \cup H_t) = (H_r \cap H_s) \cup (H_r \cap H_t) = H_{(r\oplus s)\otimes(r\oplus t)}$ and we call this header H'. Also here we pick variables as follows: for $\{r\}$ we pick a, for $\{s\}$ we pick b, for $\{t\}$ we pick c and for $\{r,s,t\}$ we pick e. Note that because of the assumption there are no variables for $\{r,s\}$, $\{s,t\}$ and $\{r,t\}$. Summarizing we have the headers $H_r = \{a,e\}$, $H_s = \{b,e\}$, $H_t = \{c,e\}$ and $H' = \{e\}$. Then, for every tuple $x \in \mathcal{T}_{\{a,e\}}$ it holds that

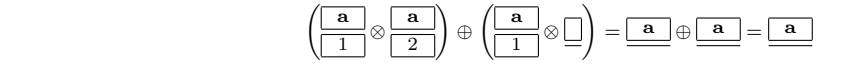
$$
\begin{aligned}
& x \in B_{r\oplus(s\otimes t)} \\
\Leftrightarrow\ & x \in B_r[e] \vee x \in B_{s\otimes t}[e] \\
\Leftrightarrow\ & x \in B_r[e] \vee (x \in B_s[e] \wedge x \in B_s[e]) \\
\Leftrightarrow\ & (x \in B_r[e] \vee x \in B_s[e]) \wedge (x \in B_r[e] \vee x \in B_t[e]) \\
\Leftrightarrow\ & x \in B_{r\oplus s} \wedge x \in B_{r\oplus t} \\
\Leftrightarrow\ & x \in B_{(r\oplus s)\otimes(r\oplus t)}.
\end{aligned}
$$

Again is it not hard to see how this proof can be generalized for arbitrary headers H_r, H_s and H_t that satisfy the assumption.

The conditions on the headers of the relations can also be shown to be necessary in the sense that for arbitrary H_r, H_s and H_t that does not satisfy it there are instances r, s and t with these headers for which distribution does not hold.

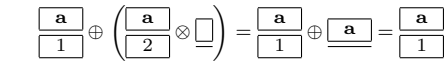
Consider the condition $H_r \cap H_s = H_r \cap H_t$ for the distribution of $\otimes$ over $\oplus$. A simple counterexample for an arbitrary attribute a and $H_r = \{a\}$, $H_s = \{a\}$ and $H_t = \emptyset$ is:

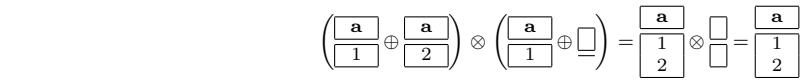
$$
\begin{array}{|c|}\hline \mathbf{a} \\ \hline 1 \\ \hline \end{array} \otimes \left(\begin{array}{|c|}\hline \mathbf{a} \\ \hline 2 \\ \hline \end{array} \oplus \begin{array}{|c|}\hline \ \\ \hline \ \\ \hline \end{array} \right) = \begin{array}{|c|}\hline \mathbf{a} \\ \hline 1 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline \ \\ \hline \ \\ \hline \end{array} = \begin{array}{|c|}\hline \mathbf{a} \\ \hline 1 \\ \hline \end{array}
$$



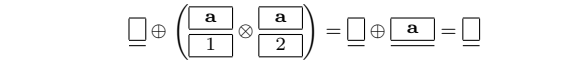
It is not hard to see how this counterexample can be extended by adding arbitrary attributes other than a to H_r, H_s and H_t and extending the tuples where necessary with a specific value for each new attribute. By symmetry there is a similar counterexample for the headers $H_r = \{a\}$, $H_s = \emptyset$ and $H_t = \{a\}$, and also this can be extended in the same way. This way we can construct counterexamples for all H_r, H_s and H_t such that $H_r \cap H_s \neq H_r \cap H_t$.

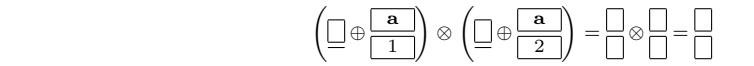
We can make a similar argument for the condition $H_r \cap H_s = H_r \cap H_t = H_s \cap H_t$ for the distribution of $\oplus$ over $\otimes$. We do this by showing first that there are simple counterexamples for each of the cases: (1) $H_r = \{a\}$, $H_s = \{a\}$ and $H_t = \emptyset$, (2) $H_r = \{a\}$, $H_s = \emptyset$ and $H_t = \{a\}$ and (3) $H_r = \emptyset$, $H_s = \{a\}$ and $H_t = \{a\}$. For case (1) $H_r \cap H_s \neq H_r \cap H_t$ we have the counterexample:





For case (2) $H_r = \{a\}$, $H_s = \emptyset$ and $H_t = \{a\}$ we have the same counterexample but s and t swapped. Finally for case (3) $H_r = \emptyset$, $H_s = \{a\}$ and $H_t = \{a\}$ we have the counterexample:





As before it is not hard to see how these counterexample can be extended by adding arbitrary attributes other than a to H_r, H_s and H_t, and this way we can construct counterexamples for all H_r, H_s and H_t such that $H_r \cap H_s \neq H_r \cap H_t$, $H_r \cap H_s \neq H_s \cap H_t$ or $H_r \cap H_t \neq H_s \cap H_t$.

4 Empty Relations

As discussed in the previous section the laws of distribution apply only under certain conditions. Such behavior can be axiomatized by introducing semi-equations that consist of a premise containing a set of equations and a conclusion containing an equation. This requires that the discussed conditions over headers of relations can be expressed in equations in our algebra. For this purpose we introduce the *header function*, whose application to a relation r is denoted as $\widehat{r}$, which is defined such that $\widehat{r} = (H_r, \emptyset)$ if $r = (H_r, B_r)$. This operation lets us express for example that r and s have the same headers by the equation $\widehat{r} = \widehat{s}$, or that the intersection of the headers of r and s is empty by $\widehat{r} \oplus \widehat{s} = \widehat{1}$.

We can now algebraically formulate the distribution laws:

$$\frac{\widehat{r} \oplus \widehat{s} = \widehat{r} \oplus \widehat{t}}{r \otimes (s \oplus t) = (r \otimes s) \oplus (r \otimes t)} \quad (11) \qquad \frac{\widehat{r} \oplus \widehat{s} = \widehat{r} \oplus \widehat{t} \qquad \widehat{r} \oplus \widehat{t} = \widehat{s} \oplus \widehat{t}}{r \oplus (s \otimes t) = (r \oplus s) \otimes (r \oplus t)} \quad (12)$$

The restriction of the Relational Lattice to the empty relations is a distributive lattice:

$$\widehat{r} \otimes (\widehat{s} \oplus \widehat{t}) = (\widehat{r} \otimes \widehat{s}) \oplus (\widehat{r} \otimes \widehat{t}) \quad (13) \qquad \widehat{r} \oplus (\widehat{s} \otimes \widehat{t}) = (\widehat{r} \oplus \widehat{s}) \otimes (\widehat{r} \oplus \widehat{t}) \quad (14)$$

Moreover there is another type of distribution that is not covered by the preceding rules, namely the fact that projections, i.e., inner unions with empty relations, distribute over a natural join if they select all attributes that the join operands have in common:

$$\frac{\widehat{r} \oplus (\widehat{s} \oplus \widehat{t}) = \widehat{s} \oplus \widehat{t}}{\widehat{r} \oplus (s \otimes t) = (\widehat{r} \oplus s) \otimes (\widehat{r} \oplus t)} \quad (15)$$

Applying the header function is equal to taking the natural join with $\widehat{1}$:

$$\widehat{1} \otimes r \quad = \quad \widehat{r} \quad (16)$$

Note that the final identity tells us that it would have been sufficient to only add the constant $\widehat{1}$ to the signature of the algebra, but we will continue to use the header function for notational compactness.

5 Equality Relations

In order to be able to express queries that compare values in different attributes we introduce an *equality relation function* that given a relation produces a relation with the same header that contains all tuples over that header with the same domain value for every attribute. The application of this function to a relation r is denoted as $\widetilde{r}$ and defined such that $\widetilde{r} = (H_r, \{\{(a, v) \mid a \in H_r\} \mid v \in \mathcal{D}\})$ if $r = (H_r, B_r)$. For example, if $H_r = \{a, b, c, d\}$ then $\widetilde{r}$ is an infinite relation:

a	**b**	**c**	**d**
0	0	0	0
1	1	1	1
2	2	2	2
⋮	⋮	⋮	⋮

Finally we add as constants *domain relations* for each attribute name $a \in \mathcal{A}$ which are denoted as a and have as their semantics the relation $(\{a\}, \{\{(a, v)\} \mid v \in \mathcal{D}\})$, i.e., the unary relation with header $\{a\}$ that contains every value in the domain $\mathcal{D}$.

The following rules show the interactions between the header function, the equality relation function and the constants.

$$\widetilde{\widetilde{r}} \quad = \quad \widetilde{r} \qquad (17) \qquad\qquad \widetilde{a} \quad = \quad a \qquad (19)$$

$$\widehat{\widetilde{r}} \quad = \quad \widehat{r} \qquad (18) \qquad\qquad \widetilde{1} \quad = \quad 1 \qquad (20)$$

The following rule states that the equality relation function distributes over the natural join if there is at least on common attribute in the header of the join operands.

$$\frac{\widehat{a} \oplus \widehat{r} = \widehat{a} \qquad \widehat{a} \oplus \widehat{s} = \widehat{a}}{\widetilde{r \otimes s} = \widetilde{r} \otimes \widetilde{s}} \tag{21}$$

If the header of a relation r contains attributes a then we can take the natural join with the domain relation a without changing the relation:

$$\frac{\widehat{a} \oplus \widehat{r} = \widehat{a}}{r \otimes a = r} \tag{22}$$

The inner union of an equality relation and an empty relation is identical to an equality relation over the inner union of both relations:

$$\widetilde{r} \oplus \widehat{s} \quad = \quad \widetilde{r \oplus s} \tag{23}$$

The inner union of two distinct domain relations is identical to $(\emptyset, \{\emptyset\})$, i.e., the relation with the empty header that contains the empty tuple:

$$a_i \oplus a_j \quad = \quad 1 \qquad \text{for all } a_i \neq a_j \tag{24}$$

The final rule captures the intuition that if we know that in a relation several attributes, say a and b, have equal values in every tuple, and one of these attributes is projected away then we can always restore it by taking a natural join with $\widetilde{a \otimes b}$. The rule states this by considering an expression of the form $((r \otimes \widetilde{s}) \oplus \widehat{t}) \otimes \widetilde{s}$ where $r \otimes \widetilde{s}$ is the relation for which we know that the attributes in the header of s are equal. This relation is projected on the header of t and after that the attributes in the header of s are all restored by taking the natural join with $\widetilde{s}$. The rule says that we can replace t with u that has a smaller header as long as (1) the headers of s and u share at least one attribute and (2) the combined headers of s and u are equal to the header of t:

$$\frac{\widehat{a} \oplus \widehat{s} = \widehat{a} \qquad \widehat{a} \oplus \widehat{u} = \widehat{a} \qquad \widehat{s} \otimes \widehat{u} = \widehat{t}}{((r \otimes \widetilde{s}) \oplus \widehat{t}) \otimes \widetilde{s} = ((r \otimes \widetilde{s}) \oplus \widehat{u}) \otimes \widetilde{s}} \tag{25}$$

This concludes the presentation of the Relational Lattice and the associated set of inference rules. Summarizing we have presented an algebra $\langle \mathcal{R}, \otimes, \oplus, 1, \widehat{\cdot}, \widetilde{\cdot}, a \rangle_{a \in \mathcal{A}}$ where $\mathcal{R}$ is the set of relations, $\otimes : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ the natural join, $\oplus : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ the inner union, 1 the relation with the empty header containing a single empty tuple, $\widehat{\cdot} : \mathcal{R} \to \mathcal{R}$ the header function that maps relations to the empty relation with the same header, $\widetilde{\cdot} : \mathcal{R} \to \mathcal{R}$ the equality relation function that maps relations to the equality relation with the same header and a for each attribute $a \in \mathcal{A}$ that represents the domain relation with one column a that contains all values in the domain.

6 Expressive Power

In this section we show that the Relational Lattice can express all queries in SPJRU, i.e., all queries that can be expressed in the relational algebra with the selection, the projection, the natural join, the rename and the union. We show that each of these operators can be simulated in the Relational Lattice:

Selection A selection $\sigma_{a=b}(r)$ can be expressed as $r \otimes \widetilde{s}$ where $s = a \otimes b$ assuming that both a and b are in the header of r.

Projection A projection $\pi_{a,b,\ldots,d}(r)$ can be expressed as $r \oplus \widehat{h}$ where $h = a \otimes b \otimes \ldots \otimes d$ assuming that all $a, b, \ldots, d$ are in the header of r.

Join The natural join $r \bowtie s$ can be expressed as $r \otimes s$.

Rename The rename $\rho_{a \mapsto e}(r)$ can be expressed as $(r \otimes \widetilde{s}) \oplus \widehat{h}$ where $s = a \otimes b$ and $h = b \otimes c \otimes d \otimes e$ assuming that the header of r is $\{a, b, c, d\}$.

Union The union $r \cup s$ can be expressed as $r \oplus s$ assuming that r and s have the same headers.

This particular class of queries is also known as UCQ, i.e., Unions of Conjunctive Queries. Observe that the Relational Lattice can also express empty queries, i.e., queries that always return an empty result over a certain header and equality relations over a certain header. In fact, as will be shown later on, it can express exactly all unions of conjunctive queries, empty queries and equality relations.

7 Completeness of the Rules

The rules 1 - 21 can be interpreted as inference rules that allow us to derive extra equations from a given set of equations. These equations are of the form $e_1 = e_2$ where e_1 and e_2 are expressions constructed from operators and constants in the algebra and a postulated countably infinite set of variable names $\mathcal{V}$ containing r, s, t, u, et cetera. In the following we will use the same symbols to represent the symbols in the equations and the operations they represent, e.g, $r \otimes s$ can both represent the expression itself or the result of the expression. The syntax of the expressions in the equations is then formally defined as follows:

$$E_0 \quad ::= \quad \mathcal{V} \mid (E_0 \otimes E_0) \mid (E_0 \oplus E_0) \mid 1 \mid \widehat{E_0} \mid \widetilde{E_0} \mid \mathcal{A}.$$

This raises the question whether as such they are complete, i.e., whether all equations that logically follow from a certain set of equations indeed can be derived. As given the rules are clearly incomplete and cannot derive for example that $r \otimes (s \oplus t) = r \otimes (t \oplus s)$ even though we can derive $s \oplus t = t \oplus s$. Therefore we extend the set of inference rules with the usual set of rules for equational reasoning that are known as Birkhoff's rules [Birkhoff, 1935]. In these rules the e_i range over expressions in the algebra, r over variables in the expressions,

$e_i[r/e_j]$ denotes the result of replacing all occurrences of the variable r in e_i with e_j, and f ranges over any function in the algebra.

$$\frac{e_1 = e_2}{e_1[r/e_3] = e_2[r/e_3]} \quad (26) \qquad \frac{e_1 = e_2}{e_2 = e_1} \quad (27) \qquad \frac{e_1 = e'_1 \quad \dots \quad e_n = e'_n}{f(e_1, \dots, e_n) = f(e'_1, \dots, e'_n)} \quad (28)$$

$$\frac{e_1 = e_2 \quad e_2 = e_3}{e_1 = e_3} \quad (29) \qquad e = e \quad (30)$$

In our proofs we will usually not mention these rules but use them implicitly by assuming that if we can rewrite an expression e_1 to e_2 using inferred equations then we can infer the equation $e_1 = e_2$ and vice versa.

We now define the notion of derivation of an equation from a set of equations S as follows. We say that $e_1 = e_2$ *derives from* S, denoted as $S \vdash e_1 = e_2$, if the equation $e_1 = e_2$ is in the smallest superset of S that is closed under the application of the inference rules 1 - 21 and Birkhoff's rules 26 - 30.

Next to a syntactical notion of entailment we also define a semantical notion of entailment. For this we define a model as a function $M : \mathcal{V} \to \mathcal{R}$ that maps each variable to a relation. The result of the evaluation of an expression e given such a model M is denoted as $[\![e]\!]_M$ and defined such that

- $[\![r]\!]_M = M(r)$ for each variable $r \in \mathcal{V}$
- $[\![e_1 \otimes e_2]\!]_M = [\![e_1]\!]_M \otimes [\![e_2]\!]_M$
- $[\![e_1 \oplus e_2]\!]_M = [\![e_1]\!]_M \oplus [\![e_2]\!]_M$
- $[\![1]\!]_M = (\emptyset, \{\emptyset\})$
- $[\![\widehat{e}]\!]_M = \widehat{[\![e]\!]_M}$
- $[\![\widetilde{e}]\!]_M = \widetilde{[\![e]\!]_M}$
- $[\![a]\!]_M = (\{a\}, \{\{(a, v)\} \mid v \in \mathcal{D}\})$

We then say that a model M satisfies an equation $e_1 = e_2$, denoted as $M \models e_1 = e_2$, if $[\![e_1]\!]_M = [\![e_2]\!]_M$. We say that a set of equations S *entails* an equation $e_1 = e_2$, denoted as $E \models e_1 = e_2$, if $M \models e_1 = e_2$ for all models M that satisfy all equations in S.

If we assume that the equations over which we reason are queries over a database then the used variables will represent relations in the database. It is therefore reasonable to assume that the headers of these relations are known and there is a function $h : \mathcal{V} \to 2^{\mathcal{A}}$ that gives a header for each variable. We will model this in our inference mechanism with equations of the form $\widehat{r} = \widehat{1}$ if $h(r) = \emptyset$ or of the form $\widehat{r} = \widehat{a_1} \otimes \ldots \otimes \widehat{a_n}$ if $h(r) = \{a_1, \ldots, a_n\}$, which we will call *typing equations*.

We are now ready to formulate the main conjecture of this paper:

Conjecture 1. *Given an equation $e_1 = e_2$ and a set of equations S that contains typing equations, exactly one for each variable in $e_1 = e_2$, then $S \vdash e_1 = e_2$ iff $S \models e_1 = e_2$.*

In the remainder of this paper we will give a sketch of the proof for this conjecture. Since the *only-if* side of the theorem follows from the preceding discussion of the rules, we will discuss here the *if* side, i.e., the completeness of the rules. This proof proceeds in the following steps:

Making headers explicit In this step it is shown that all expressions of the form $\widehat{e}$ and $\widetilde{e}$ can ben rewritten such that e becomes 1 or of the form $a_1 \otimes \ldots \otimes a_n$.

Making empty and nonempty queries explicit In this step all expressions that represent empty queries, i.e., queries that always return a relation with an empty body, are rewritten to the form $\widehat{e}$. This allows us to detect if two expressions both represent empty queries, and since e can be rewritten to an expression explicitly representing the header and we rewrite such expression to each other if the represent the same header, we are complete for empty queries. So we assume in the rest of hte proof that we are dealing with non-empty queries.

The equal header union normal form In this step all expressions are rewritten such that all inner unions that do not express a projection, i.e., are not of the form $e_1 \oplus \widehat{e_2}$ are between expressions that return relations with the same header.

The inner union normal form In this step all inner unions that do not express a projection are lifted to top level such that we get an expression of the form $e_1 \oplus \ldots \oplus e_n$ with all clauses e_i returning a result with the same header and all inner unions in the clauses e_i representing projections.

The conjunctive query normal form The clauses in the inner union normal form represent a slightly generalized type of conjunctive queries that also allow unsafe queries such as $\{(a = x, b = y, c = y) \mid p(d = x, e = x) \wedge q(f = x)\}$. We define a corresponding normal form that corresponds to the tableau and splits the query into $(e_1 \otimes \ldots e_n \otimes e_b) \oplus \widehat{e_h}$ where $\widehat{e_h}$ represents the projection on the final header, in this case $e_h = a \otimes b \otimes c$, e_b gives the binding of the variables in the final header, in this case $e_b = \widetilde{(a \otimes x)} \otimes \widetilde{(b \otimes y)} \otimes \widetilde{(c \otimes y)}$, and finally each e_i represents the atoms in the tail, in this case $e_1 = (p \otimes \widetilde{(d \otimes x)} \otimes \widetilde{(e \otimes x)}) \oplus \widehat{x}$ and $e_2 = (q \otimes \widetilde{(f \otimes x)}) \oplus \widehat{x}$. The attribute names that are in this normal form not used as headers of atoms or the final header, such as x, y and z in the example, will be called *auxiliary attribute names*. In this step we show that each clause of the union normal form can be rewritten to that normal form.

The subsumption-free normal form This step rests on the basic property of conjunctive queries that one subsumes the other iff there is a homomorphism that maps the variables of the tableau of the first to the variables of the tableau of the second. This property also holds for our slightly generalized class of conjunctive queries. This can be used to show that we can

rewrite a clause e from the conjunctive query normal form to $e \oplus e'$ where e' is the same as e except that some of the auxiliary attribute names have been renamed to other auxiliary attribute names, possibly renaming two different attribute names to the same new attribute name. This allows us to rewrite to a more strict normal form where no two clauses subsume each other.

Deciding equivalence Another property of conjunctive queries, closely connected to the previous one, is that if a conjunctive query is subsumed by a finite union of conjunctive queries then it must be subsumed by at least one of the conjunctive queries in the union. Also this property can be shown to hold for the class of queries we are considering. It follows that if a finite union of such conjunctive queries is equal to another finite union of such queries and there is no subsumption in each of the unions then both unions contain the same set of conjunctive queries. By the homomorphism property of conjunctive queries it holds that they can be only equivalent if there is in isomorphism from the variables in the tableau of one query to the variables in the tableau of the other. In terms of the conjunctive query normal form this means that two clauses are equivalent iff one is the same as the other up to the renaming of the auxiliary attribute names. It can be show that such renaming can indeed be achieved by rewriting.

8 Conclusion

In this paper we have presented the Relational Lattice along with a set of algebraic semi-equations that are conjectured to be complete for deciding the equivalence of two expressions if we can assume certain equations that makes the headers of the variables in the expression explicit. Apart from actually proving the conjecture there are many other interesting open problems that can be studied.

One type of open problem is completeness for subsets of the algebra that allow perhaps simpler axiomatizations because we are at a higher abstraction level. For example, for the algebra $\langle \mathcal{R}, \otimes, \oplus, 1 \rangle$ we can ask if the lattice laws presented in Section 2 are complete. And for the algebra $\langle \mathcal{R}, \otimes, \oplus, 1, \widehat{\cdot} \rangle$ we can ask if these laws extended with the laws in Section 4 are complete. Finally we can ask if for the algebra $\langle \mathcal{R}, \otimes, \oplus, 1, \widehat{\cdot}, \widetilde{\cdot} \rangle$ the rules that do not refer to domain relations are complete, i.e., can we reason without every referring to concrete attribute names if the query also doesn't do so?

Other interesting questions concern stronger notions of completeness of the inferences rules. As presented here they allow us to derive equations from arbitrary sets of equations. This raises the question if they are complete for such arbitrary sets, i.e., if every equation that logically follows from that set can indeed be derived. Completeness in this sense would make the inference rules interesting for reasoning over data dependencies that can be formulated as equations over algebra expressions, and reasoning over equivalency of query

expressions in the context of certain data dependencies.

Note that although the Relational Lattice does not have a complement or difference operator we can simulate the proposition $r = s - t$ with the following set of equations

$$\begin{aligned} r \otimes t &= \widehat{r} \otimes \widehat{t} \\ r \oplus t &= s \oplus t \end{aligned}$$

if we assume r, s and t have the same header. As a consequence an axiomatization of the Relational Lattice that is complete in the stronger sense would in fact be an axiomatization of all dependencies expressible in first order logic.

Acknowledgement: The author would like to thank *Vadim Tropaskho* and *Marshall Spight* for introducing him to the Relational Lattice and discussing it in the usenet group `comp.databases.theory`, and also *Maarten Marx*, *Balder ten Cate* and *Tadeusz Litak* for additional discussions and help with the algebraic approach.

References

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases.* Addison-Wesley.

[Birkhoff, 1935] Birkhoff, G. (1935). On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Date and Darwen, 2000] Date, C. J. and Darwen, H. (2000). *Foundation for Future Database Systems: The Third Manifesto.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Düntsch and Mikulás, 2007] Düntsch, I. and Mikulás, S. (2007). Cylindric structures and dependencies in relational databases. `http://www.cosc.brocku.ca/~duentsch/archive/dbcyl.pdf`.

[Henkin et al., 1971] Henkin, L., Monk, J. D., and Tarski, A. (1971). *Cylindric Algebras, Part I.* North-Holland, Amsterdam.

[Henkin et al., 1985] Henkin, L., Monk, J. D., and Tarski, A. (1985). *Cylindric Algebras, Part II.* North-Holland, Amsterdam.

[Imielinski and Lipski, 1982] Imielinski, T. and Lipski, Jr., W. (1982). The relational model of data and cylindrical algebras. In *PODS '82: Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 170–170, New York, NY, USA. ACM Press.

[Spight and Tropashko, 2006] Spight, M. and Tropashko, V. (2006). First steps in relational lattice. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0603044.

[Tropashko, 2005] Tropashko, V. (2005). Relational algebra as non-distributive lattice. http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0501053.

[Yannakakis and Papadimitriou, 1982] Yannakakis, M. and Papadimitriou, C. H. (1982). Algebraic dependencies. *Journal of Computer and System Sciences*, 25(2):2–41.

Jan Paredaens, profiel van "de prof"

Paul De Bra

In een ver verleden heb ik wel eens verhalen geschreven. Maar eigenlijk ben ik een visueel persoon, die mijn vele ervaringen met Jan dan ook beter visueel kan uitdrukken. Ik gebruik hiervoor veelvuldig mijn "PhD," ook bekend als "Push here Dummy" (of digitale camera).

In 1981 heb ik in Jan Paredaens de voor mij ideale promotor gevonden, met de aanpak die ik zelf ook tracht toe te passen: faciliteren, discussiëren, en eten. Dit wil ik aan de hand van enkele beelden illustreren.

Hoe creëert ge een groepsgevoel: door de mensen bij elkaar te brengen voor een activiteit waarbij hun gevoeligste plek gestreeld wordt: de maag. De legendarische ADREM activiteiten zijn vooral legendarisch vanwege locatie en eten. De jaarlijkse lunch is dan ook een onvergetelijk evenement, waarvan ik moet toegeven dat ik het echt een keer vergeten ben...

Discussiëren vergt een ongedwongen sfeer. En Jan is de meester in het creëren van de ideale omstandigheden hiervoor: een tafel, wat te eten, en vooral veel tijd. Nederlanders kunnen niet begrijpen dat een lunch drie uur kan duren. Maar de nieuwe ADREM lunch in "het Forum" mist wel iets: hij duurt een uur te kort.

Workshops organiseren *in* de universiteit was nooit iets voor Jan. Een activiteit op een bijzondere locatie blijft voor altijd in de deelnemers hun geheugen gegrift. Na een lange periode van stilte heeft Jan de “Database Dag” nieuw leven ingeblazen. En sindsdien is duidelijk geworden dat een bijzondere locatie de sleutel is tot het succes.

De Database Dag in Antwerpen was origineel, maar de echte “plek” van Jan ligt elders. De jaarlijkse GOOD dag is altijd bijzonder, altijd verrassend, en bijna altijd op dezelfde locatie. Het is ook bijna niet voor te stellen dat het anders zou zijn. . .

En is de workshop in het gebouw? Als het kan dan liever niet. “Bugspray” is eigenlijk wel een verplicht attribuut bij de GOOD dag, net als de Notelaartaart natuurlijk. Bij deze beelden moet ik ook terugdenken aan de “oefeningen formele talen” die ik aan de UIA heb verzorgd, buiten op het gras in plaats van binnen in een veel te heet leslokaal.

Heel wat mensen vinden dat vaste patronen saai zijn. Vele activiteiten van Jan volgen een vast patroon. Maar ik heb nog nooit iemand horen klagen dat ze saai waren. En elk jaar zijn er ook een paar nieuwelingen die moeten leren wat “paling in het groen” betekent, en waarbij soms wat overtuigingskracht nodig is om hen deze lekkernij te doen bestellen.

Wat mij bij Jan van in het begin heeft aangesproken is dat bij hem het werken altijd een klein beetje feest is. Dus wie van feesten houdt komt vanzelf aan zijn trekken. Wellicht verklaart dit ook het grote succes dat Jan altijd heeft gehad met zijn doctoraat-studenten. Wie graag werkt, werkt hard en wie hard werkt komt er wel. Ik zie collega's om mij heen die met strakke hand en heel intensief proberen om hun studenten tot aan het doctoraat te begeleiden. Maar ik zie Jan als voorbeeld. Het enige wat de studenten echt nodig hebben is plezier in hun werk en hun omgeving, en daarnaast een luisterend oor dat bereid is tot discussie en overleg, liefst in een ongedwongen sfeer. Zo hoop ik Jan nog vele wetenschappelijke kleinkinderen te kunnen bezorgen.

Due o tre cose che mi ricordo di un viaggio a Vinci

Bart Kuijpers, *Università di Hasselt*

Caro Jan, in occasione del tuo 60esimo compleanno vorrei ricordarti di un viaggio in Italia che facemmo insieme più di 10 anni fa. Mi è tornato in mente non molto tempo addietro per altri motivi, ma ti dirò più avanti quali.

Era l'estate del 1996 e a San Miniato, in Toscana, era stato organizzato il primo workshop di *Logic in Databases (LID'96)*. Prendemmo un aereo per Firenze e poi un treno fino a San Miniato. Il workshop era stato organizzato in un monastero, un luogo remoto e isolato. Le nostre camere erano anguste e poco confortevoli: proprio le stanze di monaci! Mi ricordo che le presentazioni si tenevano nella chiesa del monastero. È stata probabilmente la prima (e ultima) volta che ho parlato [1]—o piuttosto, predicato, da dietro un altare—a credenti quali Jack Minker e Jeff Ullman. La conferenza era stata organizzata da Carlo Zaniolo e Dino Pedreschi. Dino era lì con una sua giovane studentessa, Chiara Renso. Nel frattempo Dino e Chiara sono diventati entrambi amici e colleghi, con i quali collaboro da qualche anno a un progetto europeo sui datamining e GIS.

Ma il motivo per cui questo viaggio mi è tornato in mente è stata la visita che era stata organizzata per quell'occasione. Si trattava di una gita alla vicina città di Vinci, luogo di nascita di Leonardo (da Vinci). La gita prevedeva la visita alla casa natia di Leonardo e a un castello nel quale erano conservate le riproduzioni in legno delle macchine progettate dallo stesso Leonardo. È stata una di queste riproduzioni ad essermi tornata in mente non molto tempo fa (nella figura 1) e l'ho cercata sul sito del museo [2].

Lascia che ti spieghi adesso cosa mi ha fatto tornare in mente il carro armato di Leonardo. L'anno scorso stavo lavorando al trattamento dell'incertezza per dati di oggetti in movimento con un mio studente, Walied Othman. E per modellizzare l'incertezza della localizzazione di oggetti in movimento abbiamo usato *perline* (beads in Inglese). Le perline si verificano quando i dati relativi agli oggetti in movimento sono noti solo in certi punti campione. Per esempio, se di un oggetto in movimento si sa che $(\mathsf{t}_0, \mathsf{x}_0, \mathsf{y}_0), (\mathsf{t}_1, \mathsf{x}_1, \mathsf{y}_1), ..., (\mathsf{t}_N, \mathsf{x}_N, \mathsf{y}_N)$, dove $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_N$, sono momenti nel tempo e $\mathsf{x}_i, \mathsf{y}_i$ coordinate spaziali, allora si usa l'interpolazione lineare per ricostruire il tragitto prodotto dall'oggetto in movimento.

Ma cè di più! Se sappiamo che tra due punti campione $(\mathsf{t}_i, \mathsf{x}_i, \mathsf{y}_i)$ e $(\mathsf{t}_{i+1}, \mathsf{x}_{i+1}, \mathsf{y}_{i+1})$ l'oggetto non può avere una velocità maggiore di v_i, allora possiamo

Figura 1: Un carro armato disegnato da Leonardo da Vinci

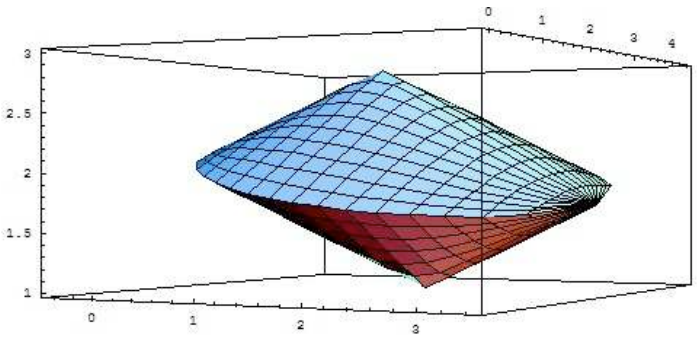


Figura 2: Una perlina disegnata in Mathematica

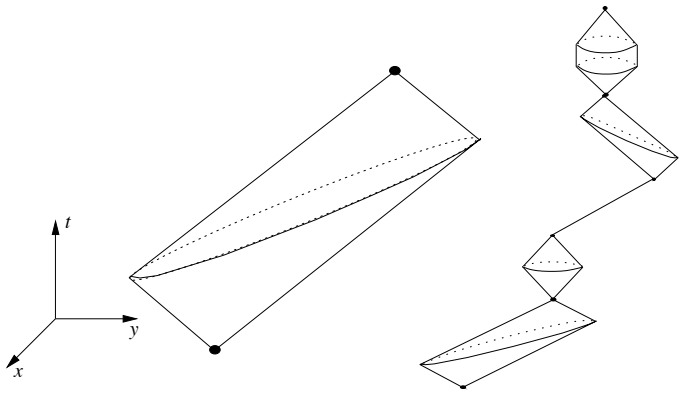


Figura 3: Una perlina e una catena

dedurre che l'oggetto in movimento non può aver lasciato l'insieme dei punti (t, x, y) date le equazioni

$$\begin{cases} (x - \mathsf{x}_i)^2 + (y - \mathsf{y}_p)^2 \leq (t - \mathsf{t}_p)^2 \mathsf{v}_i^2 \\ (x - \mathsf{x}_{i+1})^2 + (y - \mathsf{y}_{i+1})^2 \leq (\mathsf{t}_{i+1} - t)^2 \mathsf{v}_i^2 \\ \mathsf{t}_p \leq t \leq \mathsf{t}_{i+1}. \end{cases}$$

Queste equazioni definiscono la forma più semplice di una perlina. La catena di perline tra punti campione consecutivi di oggetti in movimento è stata chiamata *catenina* (*lifeline necklace* in Inglese) da Max Egenhofer [4]. Nella figura 3, sono raffiggurate entrambe (una perlina e una catena).

Visto che agli Italiani piace rivendicare per sè le invenzioni più disparate (come il telefono e gli spaghetti), sono certo che anche questa volta saranno pronti a giurare che le perline non sono state inventate da ricercatori GIS quali Dieter Pfoser nel 1999 [3] o Max Egenhofer nel 2002 [4], e neppure dal geografo del tempo T. Hägerstrand nel 1970 [5], ma da Leonardo da Vinci nel XV secolo come testimoniato da qualche suo oscuro disegno.

Disegnando le perline con Walied in Mathematica (nella figura 2), mi è tornata in mente quella macchina di Leonardo che avevamo visto a Vinci. Sembra sia un carro armato. Che scherzi può fare la memoria! Mi ricordo invece bene che dopo la visita al museo di Leonardo ci siamo seduti a un tavolo con Jeff Ullman a bere una birra godendoci il caldo sole della Toscana.

Nel frattempo, Walied, che sarà tra qualche anno il mio primo figlio e il tuo nipote ennesimo, si sta specializzando in perline [6, 7].

Dino mi ha confidato recentemente di voler organizzare presto il secondo workshop in *Logic in Databases*. Un'altra occasione per incontrarsi in Italia?

Buon compleanno, Jan!

—Bart Alto Lario, Agosto 2007

Riferimenti bibliografici

[1] B. Kuijpers, J. Paredaens, M. Smits and J. Van den Bussche. Termination Properties of Spatial Datalog Programs. In the proceedings of *Logic in Databases (LID'96)*, eds. D. Pedreschi and C. Zaniolo, volume 1154 of *Lecure Notes in Computer Science*, pages 101–116, Springer, 1996.

[2] http://www.leonet.it/comuni/vincimus/

[3] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Advances in Spatial Databases (SSD'99)*, volume 1651 of *Lecture Notes in Computer Science*, pages 111–132, 1999.

[4] K. Hornsby and M. Egenhofer. Modeling moving objects over multiple granularities. *Annals of Mathematics and Artificial Intelligence*, 36(1–2):177–194, 2002.

[5] T. Hägerstrand What about People in Regional Science? *Papers of the Regional Science Association* vol.24, 1970, pp.7-21.

[6] B. Kuijpers, W. Othman Trajectory Databases: Data Models, Uncertainty and Complete Query Languages. In *Database Theory - ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, pages 224-238, 2007.

[7] B. Kuijpers, W. Othman An analytic solution to the alibi query in the bead model for moving object data. In the proceedings of Dagstuhl seminar 07212 *Constraint Databases, Geometric Elimination and Geographic Information Systems*, 22pp., 2007.

Dear Jan, ...

Geert-Jan Houben

... On the occasion of this sixtieth birthday, thinking back on the first encounters brings me back to my studies in Eindhoven. In the brand new information systems group I became Jan's first student in Eindhoven. Jan had just started there as a part-timer and each Tuesday we would meet and discuss our work on transactions in relational databases. Impressed and inspired by the manner of coaching and collaboration, after the graduation I gladly accepted his invitation to become a PhD student. While I was stationed in Eindhoven, we would meet regularly in Eindhoven and in Antwerp. We would get more students interested in the field of databases and I was privileged to quickly move into the role of coach and learn to coach those students myself following Jan's example. From the very beginning I also remember my first trips as a PhD student. The very first one was to participate in ICDT in Rome. It meant staying over at Jan's house the night before, forcing Harry and Tim to give up one room of theirs, and then leaving from Zaventem early in the morning. It was also my first trip with some of Jan's other students, like Dirk, Marc and Paul, so a good introduction into Jan's "family" and a good introduction to the concept of academic conference. Since I was located in Eindhoven, I also started some local collaborations there and started to work as well with Kees van Hee in areas that were slightly distant from Jan's line of research, but a major line of research continued in the field of databases. One of the next conferences I went to was in Chile. I remember than after finishing the work and writing a report on it, Jan picked a conference where we could submit this as a paper and I could do my first presentation. It resulted in my first trip outside Europe, to a country that at the time was in a politically non-trivial situation, and a country that thought that a nice start to a big conference was a day full of activities in Spanish. With just my "holiday Spanish" this looked to cause a disaster, but after the Monday introductions in Spanish, the rest of the week was devoted to science and in English, so both the academic and travel adventure gave me something nice to look back to. Since we were not daily working together in person due to our different locations, the joint conference visits were good opportunities to see the master at work. I remember a trip to wintery Dresden, with so much cold that the event actually had to be moved from some resort in the countryside to the offices of the university, although in those GDR-times the university wasn't exactly the most comfortable place either. The trip with all of us in Jan's car through lots of snow on West- and East-German roads was again something that added to the fun. Lots of new things I also saw when together

with Jan we went to Japan, visiting Tokyo and Kyoto. First at a conference where I presented my work and then doing a tour of some universities where Jan gave some talks on our research. Obviously it gave many new and great impressions, and after the Chile adventure I could discover that there were languages that were even harder to communicate in, so finding directions and things to eat wasn't always that easy and a true process of discovery. The general feeling throughout all these trips was one of being with a senior academic that showed me best practices in the scientific arena and in addition was a pleasant person that was well worth of being with. After our work on complex object models led to my PhD thesis, I got the opportunity to work in Eindhoven as an assistant professor, and so our contacts would be less frequent. Throughout the evolution of my own academic career, we always would meet regularly, for example at the annual GOOD-days or January "lunches" and definitely all the times Jan would come to Eindhoven to teach. Where my own research evolved into the area of information systems design and later the Web, my inspiration from applications and new technological trends would often guarantee interesting and sparkling discussions on what we could do next. I remember that when I first brought up things from the Web field like XML, they were not always getting the respect they later got in Jan's group, and so our combination of fields proved very often to be useful and inspiring for both us in determining strategy for our research lines. I experienced that very strongly when I spent a year at Jan's group. Being involved with him and his students gave me the opportunity to develop many new inspiring ideas and set up the right conditions for my own group in Eindhoven. As a consequence I feel that I have learned a lot from Jan, in many aspects of our professional life, and am very grateful to be amongst his friends.

Congratulations to your sixtieth birthday!

Why Mathematicians make good Computer Scientists *Reflections from a biased viewpoint*

Letizia Tanca
Politecnico di Milano
tanca@elet.polimi.it

Abstract

This short paper takes its moves from a kind request by Jan Van den Bussche of contributing to Jan Paredaens's Liber Amicorum, and from my thoughts on the reasons for affinity between Jan and myself. Among other, more affective and personal ones, one reason has presented itself to my mind as being of more general interest, and it concerns our common origin as mathematicians. Thus, here I will state my point that computer scientists who start from a mathematician's background are *special* :-), and say why...

1 Introduction

Among the various affective and personal affinity reasons between Jan Paredaens and myself, I believe that our common origin as mathematicians is an important one. In these few pages I try to argue that there is a particular "brand" of computer scientists, those who start from a mathematician's background. Please notice my disclaimer: this is by no means an attempt to systematize such a difficult subject as the philosophical view of the two disciplines; indeed, no two mathematicians will ever agree on a definition of mathematics, neither two computer scientists on a definition of their field. Thus, take it as just a *divertissement*, maybe for amusement, just as, when Jan and I have worked together, we have often found the time to share a good laugh.

2 Requirements for a mathematician

In this section I present some considerations on what makes a typical mathematician; some of these considerations are mine – though far from original –, and others derive from interesting readings, like [1], [2] and observations recently done while working within the association "Informatics Europe" [3].

Let us try to summarize what appeals to mathematicians as "good" mathematical thought:

- *intellectual rigour:* on the average, mathematicians think in a very intuitive way, performing "intellectual leaps" between premises and their supposed consequences. At those moments, they may seem tolerant and genial fellows but, as soon as these leaps have to be put on paper, tolerance is no longer a value, a kind of inflexibility takes its place, and our mathematician starts to apply the most implacable rigour. This attitude is very well understood by his/her fellow mathematicians, but may appear as improper, sometimes a bit preposterous, to other researchers.

- *abstraction:* your mathematician does not like subjects which are not abstract enough. What does s/he care about electric voltage or impedance, if the subject arises during a domestic discussion about adapting the plug to the power supply? The subject would only be interesting within some scientific discussion, in a totally abstract setting...

- *generality:* again, too specific matters tend to be of no interest to our average math guy: any problem has no real appeal if it is restricted to a certain application. Often, when doing research as well in everyday life, mathematicians kind of *look* for generalizations. You will often hear a mathematician say "Let's see if this fact is true in general", be it a mathematical property or just shops' closing time.

  On the other hand, in real life, a mathematician is at a loss when trying to solve an everyday-life problem, such as a leaking pipe or a silent telephone, while s/he can give splendid explanations of general questions. This also has to do with their attitude towards practical abilities: when I was twenty, I was very good at supplying my boyfriend with articulate directions about how to park his car, but could not drive a car myself!

- *depth, difficulty:* a mathematician is always looking for difficulties to be solved – easy things are simply uninteresting.

- *simplicity,economy:* the mathematician's attitute in research is to seek the simplest solution to a problem, which reminds me of the next typical trait:

- *beauty, elegance:* maybe because I am one, like most mathematicians I've always thought that simplicity is a synonym of elegance: "A mathematician, like a painter or a poet, is a maker of patterns. If his patterns are more permanent than theirs, it is because they are made with ideas... The mathematician's patterns, like the painter's or the poet's, must be beautiful; the ideas, like the colours or the words, must fit together in a harmonious way. Beauty is the first test: there is no permanent place in the world for ugly mathematics." [1]

- *and what about utility?* Here we have a fork between pure and applied mathematicians: for the former, the utility of mathematics lays mainly into its providing exposure to deep and abstract thought, or in general, to a kind of thought which has all the above listed qualities. In this sense, the same would be true for the study of many difficult languages, such as latin, german or arabic. In the study of languages, though, the requirement of rigour, which is distinctive of mathematical thought, is not fully satisfied.

  As for the utility of the mathematician's work, instead, since mathematical models are always approximate copies of the real world, pure mathematicians are not satisfied by the use of those models for practical purposes. Their impression is that they are imprecise anyway, often cumbersome, and thus lack beauty... On the other hand, an applied mathematician finds pleasure in the process of abstraction that takes place when building models, thus for the applied mathematician "utility" acquires the usual meaning of concreteness, tangibility, applicability of one's product.

3 Requirements for a computer scientist

What is computer science about, and thus what does the typical computer scientist pursue? Computer science addresses, for example, the creation of suitable models for reality: of social and business organizations, of an image, of proteins and ribonucleic acids, of a building, of a distribution network, etc.; it addresses the problem of achieving computational efficiency (by studying computational complexity), it studies computing models and computability principles (Turing or Von Neumann machines and their extensions, quantum computing...); it introduces new programming paradigms: imperative or object oriented or functional programming, plus concurrency, coordination, communication problems and so forth; and in the more *engineering oriented* meaning of the discipline, computer science builds information infrastructures: networks and distributed systems, large computer systems, or faces more modern issues like grid and self-regulating computation, etc.

Computer science is actually a science that applies to other sciences. Examples are: bioinformatics, economics and management engineering, the informatics behind multimedia and image representation, etc. As a matter of fact, computing may even drive innovation in the other sciences: you only have to consider the human genome project, protein synthesis, environmental monitoring and protection, just to mention a few; so the idea is always to *generate*, providing a positive difference in the world.

On the other hand, in analogy with maths, practising computer science feels like:

- a "mathematical game", i.e. a challenge for intellectuals, making one feel "a high IQ person".

- as above, a way of creating patterns: the patterns computer science deals with can be numerical, spatial, temporal, and even linguistic. And through

visual and written forms of expressions, computer science is connected with mathematics to develop skills for thinking clearly, strategically, critically, and creatively.

- use of models to represent and produce changes in both real and abstract contexts, use of symbolic forms to represent and analyze real-world situations and structures.
- a discipline for problem solving, through the education to abstract knowledge representation and processing.

4 Concluding remarks

This said, very synthetically, what are the differences and analogies, in my view, between Mathematicians and Computer Scientists? A computer scientist, like an applied mathematician, takes pleasure in the process of abstraction entailed by model and algorithm design, thus the similarity here exists – though only with applied mathematicians, while pure mathematicians could not care less for the application of their results.

Moreover, as said before, the computer scientist's *main vocation* is application to other sciences: the one who tries to understand the internals of another discipline to which computer science is applied surely makes a good computer scientist. Even computer theoreticians, who exist and are well considered within the community, have an eye for the real application of their results. This aspect is also shared by applied mathematicians. However, most of the latter tend to apply different branches of maths w.r.t. those used by computer scientists. Applied mathematics mainly concerns calculus and geometry: even probability has its theoretical foundations in measure theory!! Thus, often, the mathematicians who switch to computer science are among those who love discrete mathematics, algebra and logic most.

So, what is the real difference? Actually, a very distinctive aspect is that computer science is, yes, a scientific discipline, but also a technological one. Indeed, the methods of computer science are similar to those of all scientific disciplines, since modeling and verification is the typical scientific research pattern. On the other hand, the objective of modeling reality in computer science is to feed its abstract representation to a program, in order for this to act on it and generate new behaviors and tools. Thus, here we see the use of the knowledge of scientific laws (in our case of mathematics and logic) for producing new objects: this is a characteristics of technology, and of engineering in general. As a result, a computer specialist feels the usual elation of engineers at being able to create. And this is really what mathematicians learn from becoming computer scientists: whatever they produce, *it will have to run*, that is, there is only sense in something that has the ambition of working, eventually.

But then, once a mathematician has become a computer scientist, what will distinguish his/her behaviour from that of his/her colleagues?

A "former mathematician" has a kind of compulsion to systematize, formalize, whatever new "computer phenomenon" comes up... For instance, the advent of object oriented languages, or of XML, triggered a host of researchers who wouldn't accept the news without trying to understand it by modeling it in some kind of logical/formal way...

Moreover, I often like to say that there is a rough partition among computer scientists: we'll call them the *model lovers* and the *algorithm lovers*. Model lovers are those guys who, as I said before, tend to systematize whatever they come into contact with, organizing it in a nice pattern, just as Hardy said. They love declarative specifications and languages, and tend to use logics to establish the semantics of their systems. Algorithm lovers, instead, tend to be "operational" and to solve problems in a constructive rather that in a declarative way.

My experience is that former mathematicians tend to belong to the first category, but, as I said, this distinction is quite fuzzy... Don Knuth, just to give a counterexample, studied as a mathematician, and who is more *algorithm lover* than he? But then, in his case I suppose we can make an exception, since he is one of the *inventors* of computer science, and actually in [2] he just mentions the constructive versus the declarative attitude as one main difference between compouter scientists and mathematicians.

Finally, let us also include some observation on the social view of computer scientists. While most mathematicians are perceived as some kind of lone wolves, computing is more about being part of a team that requires people with many different kinds of skills; and it is also required to possess the extensive culture and open-mindedness that are needed to take part in a creative task together with other people. This is true at the working place, but also reflects in the kind of research work we do. And it can also be seen very clearly in the number of authors of computer science versus mathematics papers: the latter are mostly one-person works, while ours have more than one author. Thus, often the former mathematician is also one that was looking for a team-based conception of the research task.

Now I am at a loss for the conclusions: I have tried to show – certainly not to prove – that former mathematicians are nice computer scientists because, in a sense, they have chosen the community and its way of doing research as being more appropriate to their inner desires, and that this applies to Jan and myself. I will be contented by Jan's sharing my opinions, but, of course, also hope that the other possible readers, maybe recognising themselves or their colleagues in my description, at least find it an occasion for a smile.

References

[1] Hardy, G.H.: A Mathematician's Apology. Cambridge Univ.Press (1992)

[2] Knuth, D.E.: Computer science and its relation to mathematics. The American Mathematical Monthly **81**(4) (apr 1974) 323–343

[3] The Informatics Europe Association: Student enrollment and image of the discipline working group, www.informatics-europe.org/wiki/index.php/image (2007)

Dear Jan, ...

Gottfried Vossen
University of Münster

... Congratulations on your 60th birthday! I am going to briefly reflect on a few occasions where our paths have crossed.

I first saw you during the ICALP conference in July 1984, which you organized in Antwerp. Shortly before that I had started working with Volkert Brosda on universal relation updates, and we stumbled upon an announcement that Ron Fagin would give an invited talk there on the theory of data dependencies. Since we both lived and worked in Aachen at the time, we drove the short distance to Antwerp in the morning before the talk, sneaked into the session without having registered (yes, I finally admit it!), and quickly disappeared again after Fagin's talk. Volkert and I were well aware of your work in database theory at the time (e.g., horizontal decompositions, BP-completeness of query languages), since we had received our database education from Joachim Biskup in an excellent course he had developed in Aachen.

I don't recall exactly when we *really* met for the first time, but I recall very well our meeting in Aigen, Austria, in late September 1989 at the 1st FMLDO workshop,[1] which I attended together with a student from Kiel (Roger Schwarz) who reported on his diploma thesis, and where you introduced a young student of yours, also named Jan, who shyly asked whether it was allowed to ask questions and who started making one significant contribution after the other shortly thereafter. We also got together for a one-day workshop at Aachen around that time which I had coined the "Euregio Database Days" since it included some 20 people from Belgium, the Netherlands, and Germany.

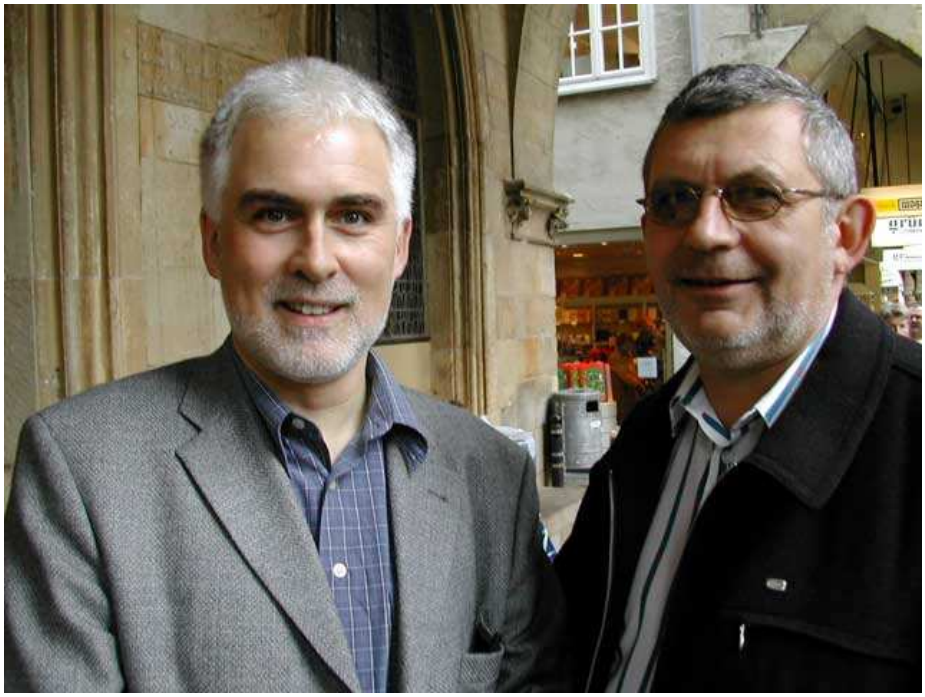
Then you agreed to become a reviewer for my habilitation procedure at the Technical University of Aachen in 1990 on database transactions (together with Walter Oberschelp and Bernd Walter, but you made my committee international). Unfortunately, on the day of the oral part of my habilitation you got stuck in Liege due to a strike of the Belgian railways.

Over the years, we have met at professional occasions numerous times, and I have always enjoyed your company, your advise, your excellent intuition about novel developments in the database field and their sustainability, but what I have always admired most is the large number of scientific children you have produced over the years and who have mostly stayed in academia as well and become successful researchers themselves. I had the great pleasure of working and writing papers with quite a few of them. One of them, Toon Calders, came

[1] http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/fmldo/fmldo89.html

with you when you visited me in Münster during October 2001. The pictures are from that visit; the first shows us in front of the Münster city hall, the second was taken at my home in the Gievenbeck suburb of Münster.

Turning 60 is a significant event in everyone's life, which is usually the reason why it is celebrated big. To cite British comedian Billy Connolly, at this age, you are "too old to die young," which is good. I am happy to see you alive and kicking at this event and that I have the pleasure to participate in the scientific meeting on its occasion. I wish you all the best for many years to come, and hope you will produce many more scientific children.



Jan Paredaens – 60

Leonid Libkin
University of Edinburgh

Looking back at my interactions with Jan over the past 18 years, I have realised that he is largely responsible for two major switches I made in my life: from math to databases, and back to math again! So now is the time for a couple of anecdotes.

From algebra to databases

We go back 18 years, to 1989, one of the most interesting years in European history. Among the most significant events of that year, at least for me, were:

- the opening of the USSR borders by Gorbachev;
- the collapse of the Eastern European communist regimes; and
- the publication of *"The Structure of the Relational Database Model"* by Jan Paredaens and co-authors [6].

I am sure there is no need to provide additional information on the first two items, but I'll say more about the third, and explain how all three are connected (at least for me).

In 1989, I was still living in Moscow, and was working on some problems related to an algebraic (lattice-theoretic, to be more precise) view of the structure of convex sets. When you deal with convex sets, one of the most common concepts you use is a convex hull, which is one of the best known examples of a closure operator. So I was working happily with closure operators until one day I was told that similar objects had arisen in an obscure (from a mathematician's point of view) subject of databases.

I had good Hungarian friends, who were quick to explain to me that databases are nothing but closure operators, although, if you wanted to be a *real* database person, you must view them as sets of implications $X \to Y$, and call elements of these sets attributes. That suited me just fine, so I decided to do some "database research". In fact it was the second time I came across databases: the first time

I had read a book in Russian, in which databases were defined as sequences of sets of homomorphism from a certain free algebra into a fixed algebra. The new definition was much nicer! I kept playing with closure operators (representing them as sets of implications, of course) and even dared to send a paper to MFDBS 1989, which was held in Hungary. Then, knowing that the iron curtain had softened to the strength of, perhaps, plastic, I applied for a permission to go to Hungary for the conference.

I still recall that the positive answer from the Soviets was a pleasant surprise, and thus I attended my first database conference, and met many researchers that I later was privileged to count among my colleagues and friends – Jan Paredaens, Georg Gottlob, Victor Vianu, Paolo Atzeni, Joachim Biskup. But perhaps the biggest revelation of MFDBS'89 was that, strangely enough, most attendees had a very different idea of what databases are!

I mentioned this to Jan during one of coffee breaks, and he promised to send me his new book, that had just appeared. A few days later, already in Moscow, I got a call that something strange called "DHL" arrived for me and I need to pick it up (I guess they had no more than a dozen DHL packages arriving to Moscow in those days). The package indeed contained Jan's book.

Having received a DHL package gave me enough courage to apply for a permanent exit visa from the USSR, and in the two months it took me to get it, I was studying Jan's book. So when I landed in the US a few months later, I was ready to start my PhD in Computer Science, this time as a real database person. Thank you, Jan!

From databases to model theory

The story does not end there. I spent the following 5-6 years happily doing databases, and even getting a few papers into SIGMOD. Then, in the summer of 1995, as I was looking at yet another flavour of the view-maintenance problem, Limsoon Wong pointed out to me that there seemed to be some interesting problems related to constraint databases.

My first reaction was negative; I was familiar with a very nice definition of the Kanellakis-Kuper-Revesz paper [3], and some fascinating open problems in the area, but had a feeling that those problems were beyond reach. Having listened to those objections, Limsoon replied that the situation was not as hopeless as I thought, and gave me a copy of the paper by Jan Paredaens, Jan Van den Bussche and Dirk Van Gucht [7], which was to appear in LICS 1995 one month later. If any subject is to attract attention, it must have an appealing definition; but to develop into a theory, it must have its own methods and interesting results. The Kanellakis-Kuper-Revesz paper introduced constraint databases, but it was not until the Paredaens-Van den Bussche-Van Gucht paper that the subject really took off the ground. The paper developed the technique

of collapse results, which turned out to be the key for developing the theory of constraint databases.

I was absolutely convinced by that paper that the problems of constraint databases can be handled, and jumped into the area (leaving a couple of view-maintenance papers unfinished). In the next few years I was busy proving collapse results and studying other properties of constraint databases. The essence of most questions related to constraint databases is the behaviour of logics over finite models embedded into an infinite structure, like $\langle \mathbb{R}, +, \cdot \rangle$, or $\langle \mathbb{N}, + \rangle$. Thus, the actual technical work consists of combining techniques of the classical, infinite, model theory, with more recently developed tools from finite model theory.

After several years of this work (which included many others – I presume quite a few of them attending this celebration), we were ready for a comprehensive survey [4]. So a decade after learning about databases from Jan's book, I had an honour of co-authoring a book with him! Furthermore, my two other books ([5] and [2]) both deal with model theory (finite, and the finite/infinite mix).

Conclusion

Largely thanks to Jan, I went from algebra to model theory via databases. As many of us know, in his *Model Theory* book [1], Keisler defined logic as what you get when you remove algebra from model theory (well, he says model theory = logic + algebra, but we all assume that + means a commutative Abelian group). So thanks to the transformations influenced so heavily by Jan, I must have picked up logic along the way, for free. Thanks, Jan, one more time!

References

[1] C.-C. Chang and H.J. Keisler. *Model Theory*. North Holland, 1990.

[2] E. Grädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer, 2007.

[3] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995. Extended abstract in *ACM Symposium on Principles of Database Systems*, pages 299–313, 1990.

[4] G. Kuper, L. Libkin and J. Paredaens, eds. *Constraint Databases*. Springer Verlag, 2000.

[5] L. Libkin. *Elements of Finite Model Theory*. Springer 2004.

[6] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht. *The Structure of the Relational Database Model*. Springer, 1989.

[7] J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order queries on finite structures over the reals. *SIAM J. Comput.* 27:1747–1763, 1998. Extended abstract in *LICS'95*.

Looking Ahead and Behind

Joachim Biskup

Fachbereich Informatik, Universität Dortmund
E-mail: biskup@ls6.informatik.uni-dortmund.de

Abstract

Since centuries, mankind has been challenged by the following problems: given the current situation, looking ahead, i.e., determining possibilities for the future, and looking behind, i.e., explaining the past. Though widely believed to be intractible, Jan Paredaens contributed essential insight to a special case identified for relational databases.

1 Problem

At any point of time, an individual or a group might want to have a look ahead in the future, or behind in the past, or even both. Take a birthday as a typical example, and still living in a decimal world and facing European traditions, assume your colleague is going to celebrate his 60th birthday. Clearly, all relatives and friends wish that the happy colleague will enjoy another 60 years, thereby looking ahead and reasoning about the actual possibilities. So the first basic problem to be investigated is the following:

> *Determine what precisely is possible in the future, given the current situation.*

Besides this, maybe after a glass of wine and starting to talk about better old times, some relatives and friends are wondering how the happy colleague happens to reach the current situation. Perhaps they know some nice details, thankfully acknowledged, but nevertheless they might feel attending a great miracle, being favored to observe the current situation. So the second basic problem to be investigated is the following:

> *Determine what precisely could have happened in the past, given the current situation.*

The sketched problems are challenging indeed, investigated over centuries. successively recognized as intractible in general. So let us try to consider special cases. As a first trial, we could instantiate the problem for a specific individual, to be more concrete, for Jan Paredaens. Well, Jan, we can't do this for you.

However, switching for a moment to another level of considerations: We would like to express our best wishes for the future and our deep thanks for what you did for us in the past.

As a second trial, we can inspect the literature for suitable special cases providing previous insight into the problems. Thankfully as we are, we start the inspection with Jan's work. And indeed, Jan successfully contributed important solutions to both problems, specialized for relational databases. In this context, the two basic problems appear in the following forms:

1. Given a relational database instance, determine which query results are producable by using the relational algebra.

2. Given a query result, determine which relational database instances could have produced it by using the relational algebra.

2 Solutions

A final solution for the first problem in the relational framework was found by Jan Paredaens around 30 years ago, as reported in [1]. Independently, Francois Bancilhon came up with the same insight, as documented in [2]. Later logicians observed that they had known the basic result before, but apparently they had not recognized it as an important contribution of database theory. Regarding more details about the pertinent relational database theory, the interested reader should inspect [1, 2] or any good textbook covering database theory.

Regarding more details about the more general version of the second problem how it could be happen, that Jan and Francois independently rediscovered such a nice result, the interested reader better exploit more intuitive methods, e.g., taking advantage of Jan's birthday celebration and drinking a class of wine together with elder colleagues, and remembering old times ... .

A partial solution for the second problem in the relational framework was elaborated by Jan Paredaens as well, this time together with some further researchers, as published in [3]. Again, regarding more details about the pertinent relational database theory, the interested reader should inspect [3].

Regarding more details about the more general version of the second problem how it could be happen that Jan once again succeeded to contribute, the interested reader will find very short historical notes in the following. In 1992, studying information security, the first author of [3] came up with the problem. He realized that a solution could be elaborated within the theory of nested relations. Appreciating Jan for years as a friendly colleague, and acknowledging his great expertise in this topic, the first author contacted Jan for cooperation. Jan kindly responded as expected, and enthousiastically started working on a solution, sharing the workload with the fourth author of [3]. After some years, having produced many versions, interfering with many other projects, and facing a rejected conference submission, all remaining issues were delegated to and solved by the third author of the final version of [3]. How could all this be

accomplished? As expected, no general answer is known. But it is a great pleasure to postulate that Jan's experience and skills, his patience and friendness essentially contributed to the final success: Jan, thank you so much for this cooperation, and all the other events we could share with you.

References

[1] J. Paredaens, On the expressive power of the relational algebra, IPL 7 (1978), pp. 44-49.

[2] F. Bancilhon, On the completeness of query languages for relational databases, In: Proc. 7th Symposium on Mathematical Foundations of Computing, Springer, 1978, pp. 112-123.

[3] J. Biskup, J. Paredaens, Th. Schwentick, J. Van den Bussche, Solving equations in the relational algebra, SIAM Journal on Computing 33,3 (2004), pp. 1052-1066.

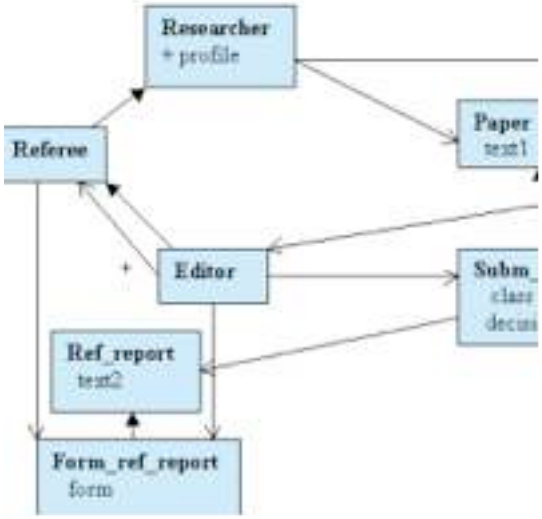
"Beste Jan," ...

Reind van de Riet

9 juni, 2007

... Zo begint een bericht dat ik je per email inmiddels heel vaak gestuurd heb. In dat bericht gaat het dan over een paper dat ik als Editor (voor Europa) van het door Elsevier uitgegeven wetenschappelijk blad Data and Knowledge Engineering (DKE), heb toegestuurd gekregen om te laten beoordelen door referees. Meestal kun jij het dan niet zelf beoordelen, omdat je het te druk hebt, maar geef je, uit je rijke ervaring en kennissenbestand een paar namen van (vaak Belgische) experts, soms je eigen leerlingen, die het paper wel kunnen beoordelen. Dat stel ik zeer op prijs. Zo dadelijk zal ik terug komen op dit zogenaamde refereeing proces, maar eerst enkele persoonlijke mededelingen.

Het zal zo'n dertig jaar geleden zijn dat we in Nederland de databaseclub oprichtten. Het was de tijd dat het relationele model in de databasewereld opgang maakte, en wij vanuit Amsterdam (de VU), Eindhoven, Twente, Utrecht en later Tilburg en Delft het idee hadden dat databaseonderzoek de moeite waard was en een eigen platform in Nederland nodig had. Al vrij vroeg kwam jij daar bij, eerst als deeltijdhoogleraar aan de TH Eindhoven (Nu Tue), maar later ook als hoogleraar uit Antwerpen. Omdat we de bijeenkomsten bij de verschillende deelnemers aan huis hielden, kwamen we zo ook in Antwerpen. Van die bijeenkomst herinner ik mij het heerlijke diner dat ons bereid was, buiten ergens op een terrasje. Toen ik in 2000 met pensioen ging was het mogelijk om editor van DKE te blijven en ik nam de gelegenheid te baat om een aantal van "mijn" referees (zo'n kleine 400) persoonlijk te bezoeken. Zo kwam ik ook in Antwerpen bij jou en je groep. Na mijn voordracht kregen we een tamelijk persoonlijk gesprek over een calamiteit die je niet lang daarvoor had moeten ondergaan; dat zal me bij blijven, en natuurlijk ook de voortreffelijke Vlaamse maaltijd die je me voorschotelde. Van die maaltijden weet ik inmiddels nog veel meer te vertellen doordat ik het voorrecht had met jou in een accreditatiecommissie to mogen zitten, die afgelopen jaar langs (bijna) alle Nederlandse Universiteiten is gegaan om aldaar de Informaticaopleidingen te beoordelen. Gedurende de vele avonden, die we als commissie na een voortreffelijk diner, doorbrachten werden kostelijke verhalen verteld; zo ook mijn verhalen over het orgelspelen. Daarover zal ik het nu niet verder hebben; nu wilde ik het hebben over je rol als referee. Ik kom hier met twee plaatjes die genomen zijn uit een verhaal/presentatie die ik over een paar dagen in Parijs zal geven over MokUM en UML. Het eerste plaatje geeft een deel van een structuur aan, waaraan in UML de namen Structure- en Object-Diagram zijn verbonden.


Researcher
+ profile
Paper
text1
Referee
Editor
Subm_
Ref_report
text2
Form_ref_report
form


PERMIT
PERMIT
REC

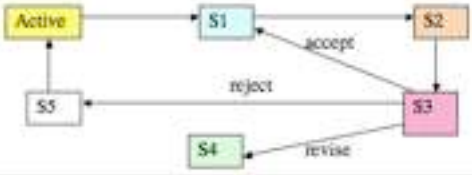
In het tweede plaatje (eigenlijk twee) zie je dat een Researcher, die een aantal Papers tot zijn beschikking heeft, er eentje kiest en naar Elsevier's website stuurt (submit). Deze plaatjes zijn geknipt uit een zogenaamd activity diagram (alle begrippen uit UML, maar alle voorbeelden met ColorX en Mokum gemaakt), zie je dat de researcher een keus maakt uit zijn papers en dat opstuurt naar Elsevier's website. Aan het eind van het proces gaat hij kijken wat de beslissing geworden is: revise, accept of reject. Het bijzondere van het voorbeeld waaruit deze plaatjes geknipt zijn is dat ze de Mokum manier weergeven om met Security en Privacy om te gaan. De referee geeft aan de Editor zijn referee report, en de Editor voegt daar een formeel gedeelte aan toe: in het attribuut form, waarin o.a. de naam van de referee vermeld staat. De researcher moet in staat zijn om de referee rapporten te lezen, maar niet de naam van de referee in het form attribuut. Wel, als de researcher kans zou zien om in het volgende stukje software, dat een vertaling is van bovenstaande stukjes uit het activity diagram, de statement 'print(... form)' te plaatsen, dan zou de Mokum compiler ingrijpen mat de foutmelding: 'access to form denied'. Dat een en ander zo is heb ik aangetoond in een artikel [1] over Security en Privacy in Cyberspace toegepast op het DKE refereeing proces, hetgeen gepubliceerd werd in het special issue DKE50, ter gelegenheid van het uitkomen van 50 volumes, alweer een paar jaar geleden en waarvan jij, als lid van de Editorial Board, ook een copy hebt gekregen. Het zal je niet moeilijk vallen om in volgende plaatjes de vorm te herkennen van een Finite State Diagram, of in UML termen een State Machine Diagram. Je ziet dat een researcher in zes verschillende toestanden kan verkeren, waarin hij komt na het lezen van verschillende boodschappen. Mocht je geïnteresseerd zijn in meer over MokUM en UML, dan moet je me maar weer eens uitnodigen. Dan kan er ook weer lekker gegeten worden. Ik wens je alle goeds, en hoop dat je je bij je pensioen (maar dat heb je met 60 nog niet bereikt) nog even goed mag voelen als ondergetekende. Nog even volhouden dus.

[1] R.P. van de Riet: The Future Refereeing Process in Cyberspace of the Data&Knowledge Engineering Journal, An Attempt in Guaranteeing Security&Privacy on three levels, Data and Knowledge Engineering, Vol 50, Nr.3, North Holland, 2004, pp. 305-339.

```
in_state(active):
next(state1), send(me, "want_to_submit_paper", ??)

in_state(state1):
on want_to_submit_paper: PA is msg.par,
send(ESS, msg="submit_paper", me.coll_of Paper[PA]),
next(state2).

in_state(state2):
on ack_from_Elsevier: next(state3).

in_state(state3):
on answer_from_Elsevier: SP is me.Subm_paper[1],
print(SP.decision), for_all ()(print(SP. coll_of
Form_ref_repor(i).text2), print(SP.coll_of(form_ref_
report(i).form)), ((SP.decision=revise), next(state4);
((SP.decision=accept), next(state1)); next(state5)).
```

```
in_state(state4):
on need_to_revise_paper: PA1 is msg.par,
PA2 is revise(PA1),
send(ESS, msg="submit_paper", PA2),
next(state2).

in_state(state5):
on need_to_rewrite_paper: PA1 is msg.par,
PA2 is rewriw(PA1),
send(somewhere, msg="submit_revised_paper", PA2),
next(active).
```

Figure xxx triggers and states for a researcher



I was lucky to have Jan around

Grzegorz Rozenberg
Universiteit Leiden, The Netherlands
University of Colorado at Boulder, USA

I was employed at the University of Antwerp (UIA) in Wilrijk from 1974 to 1979. My task was to create the computer science department (more precisely, a computer science group of the mathematics and computer science department) - hence to establish both the curriculum and the research profile. The years at UIA where both pleasant and interesting. The working environment was very nice: a small scale department, friendly and cooperative colleagues, hard working students, and well motivated Ph.D. students.

When I decided to move back to The Netherlands accepting a professorship at Leiden University the most important task was to find my successor who would continue the task of developing a computer science group with an attractive education program and worldwide research visibility. It did not take me a long time to decide that Jan Paredaens would be the most suitable person for this job. I had known Jan for some time already as both a reliable and pleasant colleague and a serious researcher with a good research record. An attractive feature of his research was a good balance of theory and applications. He was doing research on foundations of programming and theory of databases which had a genuine motivation in applications.

It was not difficult to convince my colleagues that Jan was the best candidate for the job. Consequently he was offered the position which he gladly accepted (if I remember well he was working then at the Philips Research Labs in Brussels). In retrospect it was a very good decision indeed. Jan has strengthened the existing group, extended it, and most importantly today the group enjoys a very good international research reputation which relies very much on the excellent research by him, his co-workers, and Ph.D. students.

Looking back into my Antwerp years and the years since then, I see the founding of the computer science department in Antwerp as one of my valuable contributions to the development of the computer science community. In particular, I am glad that I have chosen Jan to be my successor. I was very lucky that I knew Jan at the time when the decision had to be taken.

Dear Jan, thank you very much for all your efforts through almost 30 years to build up the computer science department in Antwerp. One's 60th birthday is an important reflection point in our lives. I think that you can really look back with satisfaction and pride both at your own scientific career, and at what you have done for the community.

I wish you still many years of a fruitful professional and happy personal life.

Grzegorz

Happy Birthday, Jan!

Victor Vianu
U.C. San Diego
vianu@cs.ucsd.edu

When thinking of various people, I like to perform the following mental experiment: close my eyes, visualize the person, and let words describing him surge out. In Jan Paredaens's case, these are words like: creative, brilliant, understated, insightful, clear-minded, innovative, inspiring, paternal, stimulating, modest, decent.

Although Jan is only eight years my senior, I have looked up to him ever since I was a graduate student, when he appeared to me as one of the Database Theory Gods. Jan has made foundational contributions to almost every classical area in database theory, including dependency theory and schema design, object-oriented databases and complex objects, graph data, and spatial databases. In the age of the Web, he has been instrumental in laying the foundations of semi-structured data, Web data, and XML query languages. I have been struck numerous times by Jan's ability to continuously renew himself as a researcher, dive into a new area and bring to bear his talent for extracting order and clarity out of chaos. His talks and articles have directly inspired my own research on several occasions. I recall for example his eye-opening invited talk "Spatial databases, the final frontier" from ICDT 1995. In characteristic manner, Jan's talk revealed the beauty of this new subject, and formulated fascinating challenges that made one immediately want to go home and do research in the area.

Jan has made many contributions to the scientific community, but perhaps the most remarkable has been to grow and nurture Belgium's outstanding database theory community, one of the strongest and most vital in the world.

On this milestone in Jan's career, I can only wish that he will continue to be a joyful companion and leading researcher in our area for many years to come. Happy Birthday, Jan!

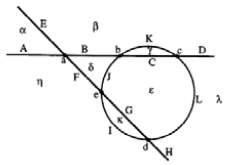
SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1747–1763, December 1998

011

FIRST-ORDER QUERIES ON FINITE STRUCTURES OVER THE REALS*

JAN PAREDAENS†, JAN VAN DEN BUSSCHE‡, AND DIRK VAN GUCHT§

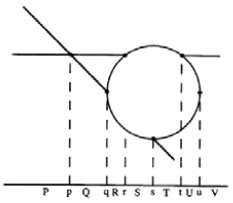
...erties of finite relational structures over the reals expressed by



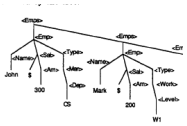
Spatial Databas

The Final Frontier

Jan Paredaens
University of Antwerp, Belgium
pareda@uia.ua.ac.be



1. *Equivalence classes in the plane induced by a circle and two line*



This tree is generated by a grammar with prod

P1 : ⟨Emps⟩ → ⟨Emp⟩⟨Emps⟩
P2 : ⟨Emp⟩ → ⟨Name⟩⟨Sal⟩⟨Type⟩
P3 : ⟨Type⟩ → ⟨Man⟩
P4 : ⟨Type⟩ → ⟨Work⟩
P5 : ⟨Man⟩ → ⟨Dep⟩
P6 : ⟨Work⟩ → ⟨Level⟩
P7 : ⟨Sal⟩ → $⟨Am⟩

Towards a Theory of Spatial Database Queries

(Extended Abstract)

Jan Paredaens*
University of
pareda@wins

Jan Van den Bussche†

Dirk Van Gucht‡

572

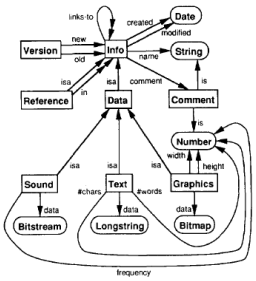
A GRAMMAR-BASED APPROACH TOWARDS UNIFYING HIERARCHICAL DATA MODELS
(extended abstract)

Marc Gyssens, Jan Paredaens, Univ. of Antwerp (UIA), B-2610 Antwerpen, Belgium
Dirk Van Gucht, Indiana Univ., Bloomington, IN 47405-4101, USA

IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 6, NO. 4, AUGUST 1994

A Graph-Oriented Object Database Model

Marc Gyssens, Jan Paredaens, Jan Van den Bussche, and Dirk Van Gucht



***Abstract*—A graph-oriented object database model (GOOD) is introduced as a theoretical basis for database systems in which manipulation as well as conceptual representation of data is transparently graph-based. In the GOOD model, the scheme as well as the instance of an object database is represented by a**

specified as graphs, queries are expressed textually and hence can become quite cumbersome.

The first graphical database end-user interfaces were developed for the relational model (e.g., QBE [29]). The earliest

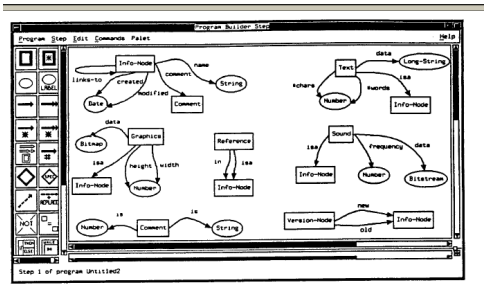


Figure 1: A GOOD database scheme.

Converting Nested Algebra Expressions
Flat Algebra Expressions

JAN PAREDAENS
University of Antwerp
and

$$h_{ii'} \equiv \exists v_{i1}[\lambda_{i1}] \cdots \exists v_{in_i}[\lambda_{in_i}] \exists v'_{i'1}[\lambda'_{i'1}] \cdots$$

$$\exists v'_{i'n'_{i'}}[\lambda'_{i'n'_{i'}}] \left(\bigwedge_{j=1}^{l_i} c_{ij} \bigwedge_{j'=1}^{l'_{i'}} c'_{i'j'} \wedge \neg \right.$$

$$\left. \neg\, g \equiv \neg\, g, \exists t[\lambda] g \equiv \left(\bigvee_{i=1}^{k} h''_i \right), \right.$$

Axiomatization of Frequent Sets

Toon Calders* and Jan Paredaens

HAPPY BIRTHDAY, DEAR JAN!

You have changed our field to the better!

Greetings, Georg

My first encounter with Jan was in 1979 when I defended my PhD thesis and he was a member of the jury. Although logic programming was not close to his own research interests, he was very supportive of my work. During the following years, I can remember some occasions where he invited me to Antwerp for giving a seminar about my work.

Our interactions got more intense and regular in 1995, when we worked together on a proposal for a *Scientific Research Network ("WOG")* that we submitted to the *Research Foundation - Flanders ("FWO-Vlaanderen")*. The network, named *Declarative Methods in Computer Science* focussed on logic programming and databases, which were the main interests of our respectively research groups. Although Jan had always a busy schedule, he was very responsive and he contributed a lot. It was very stimulating to work with him on the proposal. The proposal was approved and the network started in 1996 for a 5 year term. Currently it is in its third extension and our both groups, together with the Theoretical Computer Science group of University of Hasselt —also one of the initial participants— are still the core groups in the network. The network stimulated the interaction between our groups and we currently have some genuine collaboration at the level of projects funded by the Research Foundation - Flanders.

Jan, it has been and still is a pleasure to collaborate with you, and your example has been a stimulation to strive for excellence in research.

Maurice Bruynooghe
Department Computer Science
Katholieke Universiteit Leuven

A database full of memories

Andy Zaidman

When thinking of Jan, lots of memories come back to mind. Over the almost ten years that I've known Jan, I have filled up a complete database of memories. When 'mining' this database, one sees that there are three main associations that I have made to Jan...

The first of these associations, is Jan in his natural habitat as a professor and a lecturer. My very first recollection of Jan goes back to my second year as a computer science bachelor (or 'kandidatuur' like they called it then) student. He was the man that taught us about relational algebra, SQL, E/R diagrams and so many of the other database basics that we now all take for granted. It was also immediately clear from his classes, that Jan very much liked formal semantics and formal proofs.

A few years later I gained another perspective of Jan, more precisely when we were both part of the computer science education commission. There, I had the opportunity to get to know Jan from a very different side. Now, I wasn't seeing the lecturer, but an accomplished politician. Or maybe we should even consider him more of a diplomat, because Jan, being very much aware of what Jan wants for himself, is also able to make the right compromise at the right time in order to get what he wants, but also what the others deem essential.

A third association that I have with Jan is that we are co-authors of two papers. Even though I have never been a member of his ADReM research group, I still had the opportunity to have research discussions with him and Toon Calders, when we were exploring the boundaries of software engineering and datamining. This resulted in a technique to mine execution traces with the help of web mining.

Our research collaboration also earned Jan a place in my PhD jury in September 2006. Since then our paths have somewhat diverged, with Jan spending lots of time as a dean and me moving to Delft, but I am sure that the database will be filled with many more memories to come.

Pearls of Modelling: From Relational Databases To XML Suites

Bernhard Thalheim

Christian-Albrechts University Kiel

Devoted to the 60^{th} birthday of prof. J. Paredaens

Abstract

The theoretical foundations for modelling database structures have been extensively developed over the last decades. Structures consider, however, only one side of the modelling coin. A coin has two sides, i.e. in addition to the structure integrity constraints and their treatment must be considered. We develop a unifying approach to specification and treatment of integrity constraints and illustrate this unifying theory for two structures: object-relational models as the extension of the relational model and semi-structured models that form the basis of the internet technology. We survey pearls, nuggets and lessons learned in the school of J. Paredaens and their collaborators and highlight the approach to be taken for achieving a consistent understanding of the XML coin.

1 Introduction

Database literature and teaching is divided into at least two branches: applications [SW05] and their formal treatment on the basis of database theory [PBGG89]. The first branch uses database theory mainly on the basis of results obtained until the mid-80ies. For computer engineers, logics and algebra becomes more and more a 'Terra incognita'. There are already statements that database theory research is 'dead on its feet'[1]. However, database theory, database application formalization and database applications have gained from logics and discrete mathematics more than it is acknowledged.

1.1 Database Design and Development

The problem of database design can be stated as follows:
Design the logical and physical structure of a database in a given database management system (or for a database paradigm), so that it contains all the information required by the user and required for the efficient behavior of the whole information

[1] M. Stonebraker, ICDE, Vienna 1993

system for all users. Furthermore, specify the database application processes and the user interaction.

The implicit goals of database design are:

- to meet all the information (contextual) requirements of the entire spectrum of users in a given application area;
- to provide a "natural" and easy-to-understand structuring of the information content;
- to preserve the designers entire semantic information for a later redesign;
- to achieve all the processing requirements and also a high degree of efficiency in processing;
- to achieve logical independence of query and transaction formulation on this level;
- to provide a simple and easily to comprehend user interface family.

Over the last years database structures have extensively been discussed. Almost all open questions have been satisfactorily solved. Modelling includes, however, more tasks which are not solved at the same level. Often, modelling is considered to have at least "two sides of a coin": OPM - object-process modelling. This separation into static and dynamic aspects is correct to a certain extend. We claim, however, that modelling is far more complex. Modelling of a database application includes at least three different sides:

Structuring of a database application is concerned with representing the database structure and the corresponding static integrity constraints.

Functionality of a database application is specified on the basis of processes and dynamic integrity constraints.

Interactivity is provided by the system on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

This understanding has led to the **codesign approach** to modelling by modelling structuring, functionality and interactivity. These three aspects of modelling have both syntactic and semantic elements. If we differentiate between the syntactic and semantic elements we may compare the elements of modelling with the six sides of a dice: structure, processes, static and semantic integrity constraints, representations and stories.

Integrity constraints are used to separate "good" states or sequences of states of a database system from those which are not intended. They are used for specification of semantics of both structures and processes. Therefore, consistency of database applications can not be treated without constraints. At the same time, constraints are given by users at various levels of abstraction, with a variety of vagueness and intensions behind and on the basis of different languages.

For treatment and practical use, however, constraints must be specified in a clear and unequivocal form and language. In this case, we may translate these constraints to internal system procedures which are supporting consistency enforcement.

1.2 Storage and Representation Alternatives

The classical approach to objects is to store an object based on strong typing. Each real life thing is thus represented by a number of objects which are either coupled by the object identifier or by specific maintenance procedures. This approach has led to the variety of types. Thus, we might consider two different approaches:

Class-wise, strongly identification-based storage: Things of reality may be represented by several objects. Such choice increases maintenance costs. For this reason, we couple things under consideration and objects in the database by an injective association. Since we may be not able to identify things by their value in the database due to the complexity of the identification mechanism in real life we introduce the notion of the *object identifier* (OID) in order to cope with identification without representing the complex real-life identification. Objects can be elements of several classes. In the early days of object-orientation it has been assumed that objects belong to one and only one class. This assumption has led to a number of migration problems which have not got any satisfying solution. Their association is maintained by their object identifier.

Object-wise storage: The graph-based models which have been developed in order to simplify the object-oriented approaches [BT99] display objects by their sub-graphs, i.e. by the set of nodes associated to a certain object and the corresponding edges. This representation corresponds to the representation used in standardization.

Object-wise storage has a high redundancy which must be maintained by the system thus decreasing performance to a significant extent. Beside the performance problems such systems also suffer from low scalability and bad utilization of resources. The operating of such systems leads to lock avalanches. Any modification of data requires a recursive lock of related objects.

For these reasons, objects-wise storage is applicable only under a number of restrictions:

- The application is stable and the data structures and the supporting basic functions necessary for the application are not changed during the lifespan of the system.
- The data set is almost free of updates. Updates, insertions and deletions of data are only allowed in well-defined restricted 'zones' of the database.

A typical application area for object-wise storage are archiving or information presentation systems. Both systems have an update system underneath. We call such systems play-out system. The data are stored in the way in which they are transferred to the user. The data modification system has a play-out generator that materializes all views necessary for the play-out system.

Other applications are main-memory databases without update. The SAP database system uses a huge set of related views. The generation of the view system consumes hours of the large Oracle database system that is supporting the main-memory database. The retrieval of the system is, however, very fast for all situations which are supported by the view set. All data required in this case by the application must be stored in one of the views. Any change in the application requires the development of another view set. The main-memory database increases its size in an avalanche form.

The two implementation alternatives are already in use although more on an intuitive basis:

Object-oriented approaches: Objects are decomposed into a set of related objects. Their association is maintained on the basis of OID's or other explicit referencing mechanisms. The decomposed objects are stored in corresponding classes.

XML-based approaches: The XML description allows to use null values without notification. If a value for an object does not exist, is not known, is not applicable or cannot be obtained etc. the XML schema does not use the tag corresponding to the attribute or the component. Classes are hidden. They can be extracted by queries of the form:

```
select object
from site etc.
where component xyz exist
```

Thus, we have two storage alternatives which might be used at the same time or might be used separately:

Class-separated snowflake representation: An object is stored in several classes. Each class has a partial view on the entire object. This view is associated with the structure of the class.

Full-object representation: All data associated with the object are compiled into one object. The associations among the components of objects with other objects are based on pointers or references.

We may use the first representation for our storage engine and the second representation for out input engine and our output engine in data warehouse approaches. The input of an object leads to a generation of a new OID and to a bulk insert into several classes. The output is based on views.

The first representation leads to an object-relational storage approach which is based on the ER schema. Thus, we may apply translation techniques developed for ER schemata[Tha00].

The second representation is very useful if we want to represent an object with all its facets. For instance, an *Address* object may be presented with all its data, e.g., the geographical information, the contact information, the acquisition information etc. Another *Address* object is only instantiated by the geographical information. A third one has only contact information. We could represent these three object by XML files on the same DTD or XSchema.

We have two storage options for the second representation in object-relational databases: either to store all objects which have the same information structure in one class or to decompose the objects according to the structuring schema. Since the first option causes migration problems which are difficult to resolve and which appear whenever an object obtains more information, we prefer the second option for storing. In this case the XML representations are views on the objects stored in the database engine. The input of an object leads to a generation of a new OID and to a bulk insert into several classes. The output is based on views.

2 Structuring Models For Relational Databases

Structuring of databases is based on three interleaved and dependent parts:

Syntactics: Inductive specification of database structures based on a set of base types, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. Structural recursion is the main specification vehicle.

Semantics: Specification of admissible databases on the basis of static integrity constraints describes those database states which are considered to be legal. If structural recursion is used then a variant of hierarchical first-order predicate logics may be used for description of integrity constraints.

Pragmatics: Description of context and intension is based either on explicit reference to the enterprise model, to enterprise tasks, to enterprise policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

The rich theory developed by Jan Paredaens and his pupils has led to a deep understanding of various kinds of normalisation (vertical, horizontal and deductive) [PBGG89] of various extensions of the relational database model [GPG88], of various kinds of constraints [GP83, GP86, Par82], and of the algebra of the relational database model [HPT88].

2.1 Rigid Inductive Structures

The inductive specification of structuring is based on base types and type constructors. A base type is an algebraic structure $B = (Dom(B), Op(B), Pred(B))$ with a name, a set of values in a domain, a set of operations and a set of predicates. A class B^C on the base type is a collection of elements form $dom(B)$. Usually, B^C is required to be set. It can be list, multi-set, tree etc. Classes may be changed by applying operations. Elements of a class may be classified by the predicates.

A *type constructor* is a function from types to a new type. The constructor can be supplemented with a *selector* for retrieval (such as *Select*) and *update functions* (such as *Insert*, *Delete*, and *Update*) for value mapping from the new type to the component types or to the new type, with correctness criteria and rules for validation, with default rules, with one or more user representations, and with a physical representation or properties of the physical representation.

Typical constructors used for database definition are the *set*, *tuple*, *list* and *multiset* constructors. For instance, the set type is based on another type and uses algebra of operations such as union, intersection and complement. The retrieval function can be viewed in a straightforward manner as having a predicate parameter. The update functions such as *Insert*, *Delete* are defined as expressions of the set algebra. The user representation is using the braces $\{,\}$. The type constructors define type systems on basic data schemes, i.e. a collection of constructed data sets. In some database models, the type constructors are based on pointer semantics.

General operations on type systems can be defined by *structural recursion*. Given a types T, T' and a collection type C^T on T (e.g. set of values of type T, bags, lists) and operations such as generalized union $\cup_{C^T}$, generalized intersection $\cap_{C^T}$, and generalized empty elements $\emptyset_{C^T}$ on C^T. Given further an element h_0 on T' and two functions defined on the types

$$h_1 \quad : \quad T \rightarrow T'$$

and

$$h_2 \quad : \quad T' \times T' \rightarrow T' \quad .$$

Then we define the structural recursion by insert presentation for R^C on T as follows

$srec_{h_0,h_1,h_2}(\emptyset_{C^T}) = h_0$
$srec_{h_0,h_1,h_2}(|\{|s|\}|) = h_1(s)$ for singleton collections $|\{|s|\}|$
$srec_{h_0,h_1,h_2}(|\{|s|\}| \cup_{C^T} R^C) = h_2(h_1(s), srec_{h_0,h_1,h_2}(R^C))$
iff $|\{|s|\}| \cap_{C^T} R^C = \emptyset_{C^T}$.

All operations of the relational database model and of other declarative database models can be defined by structural recursion. Structural recursion is also limited in expressive power. Nondeterministic while tuple-generating programs (or object generating programs) cannot be expressed. We observe, however, that XML together with the co-standards does not have this property.

Another very useful modelling construct is *naming*. Each concept type and each concept class has a name. These names can be used for the definition of further types.

Static integrity constraints are specified within the universe of structures defined by the structural recursion.

Observation 1.
Hierarchical structuring of types leads to a generalized first-order predicate logics.

Observation 2.
In general, cyclic structuring leads to non-first-order logics. Structures with abstract linking are potentially cyclic.

2.2 Static Integrity Constraints

Each structure is also based on a set of implicit model-inherent integrity constraints:

Component-construction constraints are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.

Identification constraints are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups [BT99]. We shall later see that value-representability or weak-value representability lead to controllable structuring.

Acyclicity and finiteness of structuring supports axiomatization and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

Superficial structuring leads to representation of constraints through structures. In this case, implication of constraints is difficult to characterize.

Implicit model-inherent constraints belong to the performance and maintenance traps.

Integrity constraints can be specified based on the B(eeri-)V(ardi)-frame, i.e. by an implication with a formula for premises and a formula for the implication. BV-constraints do not lead to rigid limitation of expressibility. If structuring is hierarchic then BV-constraints can be specified within the first-order predicate logic. We may introduce a variety of different classes of integrity constraints defined:

Equality-generating constraints allow to generate for a set of objects from one class or from several classes equalities among these objects or components of these objects.

Object-generating constraints require the existence of another object set for a set of objects satisfying the premises.

A class $\mathcal{C}$ of integrity constraints is called *Hilbert-implication-closed* if it can be axiomatized by a finite set of bounded derivation rules and a finite set of axioms. It is well-known that the set of join dependencies is not Hilbert-implication-closed for relational structuring. However, an axiomatization exists with an unbounded rule, i.e. a rule with potentially infinite premises.

Often structures include also optional components. Let us denote the set of all components of a set $\mathcal{O}$ of objects by $compon(\mathcal{O})$ and the set of all optional components of $\mathcal{O}$ by $compon^{opt}(\mathcal{O})$. Similarly we denote the set of all components used in a constraint α by $compon(\alpha)$. Validity of constraints is either based on strong semantics requiring validity for all object sets independently on whether $compon^{opt}(\mathcal{O}) \cap compon(\mathcal{O}) \neq \emptyset$ or on weak semantics requiring validity for constraints only for those object sets $\mathcal{O}$ for which $compon^{opt}(\mathcal{O}) \cap compon(\mathcal{O}) = \emptyset$. Classical validity is based on weak semantics which has a severe disadvantage:

Observation 3.
Weak semantics leads to non-additivity of constraints for object sets $\mathcal{O}$ with $\mathcal{O}$ by $compon^{opt}(\mathcal{O}) \neq \emptyset$, i.e., it is not true in general that $\mathcal{O} \models \{\alpha_1, ..., \alpha_m\}$ is valid if and only if $\mathcal{O} \models \{\alpha_i\}$ for each constraint in $\{\alpha_1, ..., \alpha_m\}$.

Observation 4.
Strong semantics leads to non-reflexiveness or non-transitivity of constraints for object sets $\mathcal{O}$ with $\mathcal{O}$ by $compon^{opt}(\mathcal{O}) \neq \emptyset$, i.e., $\mathcal{O} \not\models \alpha \to \alpha$ for some constraints α or the validity of $\mathcal{O} \models \alpha \to \beta$ and $\mathcal{O} \models \beta \to \gamma$ does not imply $\mathcal{O} \models \alpha \to \gamma$.

Since constraint sets may be arbitrary we might ask in which cases an axiomatization exists. The derivation operator $\vdash_\Gamma$ of a deductive system Γ and the implication operator $\models$ may be understood as closure operators Φ, i.e.

(0) $\Phi^0(\Sigma) = \Sigma$
(i) $\Phi^{i+1}(\Sigma) = \{\alpha \in \mathcal{C} \cap \Phi(\Phi^i(\Sigma))\}$
(+) $\Phi^*(\Sigma) = lim_{i \to \infty} \Phi^i(\Sigma)$
for any subset Σ from a class $\mathcal{C}$ of constraints.

The closure operator Φ is called compact for a class $\mathcal{C}$ if the property $\alpha \in \Phi^*(\Sigma)$ implies the existence of a finite subset Σ' of Σ such that $\alpha \in \Phi^*(\Sigma')$. It is called closed of $\Phi^*(\Phi^*(\Sigma)) = \Phi^*(\Sigma)$ for any $\Sigma \subseteq \mathcal{C}$. The closure operator is called monotone if $\Phi^*(\Sigma) \subseteq \Phi^*(\Sigma \cup \Sigma')$. The operator is reflexive if $\alpha \in \Phi^*(\Sigma \cup \{\alpha\})$ for all formulas and subsets from $\mathcal{C}$.

Observation 5.
The implication operator $\Phi^*_\models$ is reflexive, monotone, closed and compact if and only if there exists a deductive system Γ such that Φ_Γ and $\Phi_\models$ are equivalent. If $\Phi_\models$ additionally has the inference property, the deduction property and is generalization invariant then $\Phi^*_\Gamma(\emptyset) = \Phi^*_\models(\emptyset)$.

If the deduction property fails then the axiomatization by a deductive system may be based on some obscure rules similar to those for the axiomatization of

PROLOG.

Constructors used for construction of more complex types are often used for convenience and representing a different structuring. A typical example is the application of the list constructor with the meaning of representing sets. In this case we must add an list-to-set axiom
$\forall t \in compon(o) \forall i,j (type(o.i) = type(o.j) = t \ \Rightarrow \ value(o.i) = value(o.j))$.
This axiom is often overseen and not considered.

Observation 6.

Semantics for structures defined by the list constructor and representing set must be extended by list-to-set axiom.

Since attributes are also constructed on the basis of constructors from base types we may ask whether this construction affects the definition of constraints and the axiomatizability. This question is open for most of the constraints. In [Lin03] it has, however, shown that keys and functional dependencies have a similar treatment as in the relational case. Substructures are, however, more complex and represented by the Brouwerian algebra of subcomponents.

3 Application of the Theory to Database Structuring

3.1 Pearls of Database Research Applied to Semantical Models

The entity-relationship model has been extended to the higher-order entity-relationship model (HERM)[Tha00]. HERM is a set-theoretic based, declarative model which objects are value-oriented. For this reason, object identifiers are omitted.

The entity-relationship model uses basic (or atomic) data types such as INT, $STRING$, $DATE$, etc. the null type $\perp$ (value not existing). Using type constructors for tuples, finite (multi-)sets and lists and union we construct more complex types based on standard set semantics:

$$t = l : t \mid B \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t'\} \mid \langle t' \rangle \mid [t'] \\ \mid (a_1 : t_1) \cup \cdots \cup (a_n : t_n)$$

These types will be used to describe the domains of (nested) attributes.

Attributes allow to conceptually abstract from describing values. Associated data types provide the possible values. We use a set of *names* $\mathcal{N} = \mathcal{N}_0 \cup \mathcal{N}_c$ for attributes such as *address, street, city, name, first_name, destination, trip_course* etc. Elements from $\mathcal{N}_0$ are called atomic attributes.
Each atomic attribute is associated with a atomic type $dom(A)$.
Nested attributes are inductively constructed from simpler or atomic attributes by the iteration condition:
For already constructed nested attributes $X, X_1, ..., X_n$ and new attributes $Y, Y_1, ..., Y_m$ the sequences

$Y(X_1, ..., X_n)$, $Y\{X\}$, $Y\langle X\rangle$, $Y[X]$, $Y((Y_1 : X_1) \cup \cdots \cup (Y_n : X_n))$
are tuple-valued, set-valued, list-valued, multiset-valued and union-valued nested attributes.
Associated complex types are defined by the attribute structure. In the logical calculus below we use only tuple-valued and set-valued attributes. The calculus can similarly be extended.

For all types we use set semantics based on the basic type assignment.

Entity Types (or *level-0-types*) $E = (attr(E))$ are defined by a set $attr(E)$ of nested attributes. A subset X of attributes can be used for identification. This subset is called key of E. In this case we consider only those classes which objects can be distinguished by their values on X.

Relationship Types (or *level-(i+1)-types*) $R = (comp(R), attr(R))$ are defined by a tuple $comp(R)$ of component types at levels $\leq i$ with at least one level-i-type component and a set $attr(R)$ of nested attributes. We use set semantics with expanded components under the restriction that $comp(R)$ forms a key of R. Unary relationship types with $|comp(R)| = 1$ are subtypes.

Clusters (also level-i-types) $C = C_1 \oplus \cdots \oplus C_k$ are defined by a list $\langle C_1, \ldots, C_k\rangle$ of entity or relationship types or clusters (components). The maximal level of components defines level i. We set semantics (union) or equivalent pointer semantics.

Corresponding classes of a type T are denoted by T^C. $\mathcal{R}(T)$ is the set of all classes of T. Basic type assignment is equivalent to pointer semantics with *value representability*.

The usual graphical representation of the extended ER model is a labeled directed acyclic graph. Entity types are denoted graphically by rectangles. Relationship types are graphically represented by diamonds with arrows to their components. Attributes are denoted by strings and attached to the types by lines. Key components are underlined.

A HERM scheme S is a set $\{R_1, ...R_m\}$ of types of level $0, ..., k$ which is closed, i.e. each set element has either no components (entity type) or only components from $\{R_1, ...R_m\}$.

An example of a HERM diagram is displayed in Figure 1.

A person can be customer. People are identified by their names. Customers have an additional customer identification. A store is characterized by its name and address. For each purchase a customer can buy several parts. Addresses and names can be more complex than displayed in the diagram. The diagram is used for representing types. For instance, in our example the relationship type *Purchase* is defined by

Purchase = (Customer, Store, { TimeOfPurchase, TypeOfPayment }) .

Since customers are persons purchases are identified by the name of the person and the name and the address of the store.

Based on the construction principles of the extended ER model we can introduce the HERM algebra [Tha00]. In general, for a HERM scheme S the set $Rec(S)$ of all expressions definable by structural recursion can be defined.

Since HERM database schemes are acyclic and types are strictly hierarchical

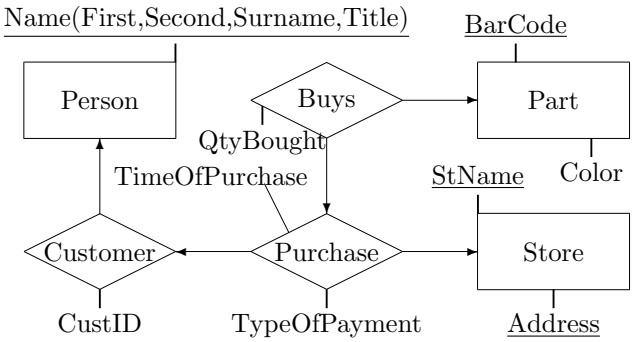


Figure 1: A Customer-Relationship Diagram

we can construct a many-sorted logical language by generalizing the concept of variables.

Given a HERM scheme S. Let $\mathcal{N}_S$ the set of all names used in S including type names. A sort is defined for each name from $\mathcal{N}_S$. The sort sets are constructed according to the type construction in which the name has been used. Entity and relationship types are associated with predicate variables. The logical language uses type expansion for representation of relationship types. We can use key-based expansion or full expansion. Full expansion uses all components of the component type. Key-based expansion uses only (primary) key components. If all names are different in S then we can use lower-case strings for variable names. If this is not the case then we use a dot notation similar to the record notation in programming languages.

For instance, the entity type *Person* is associated with the unary predicate variable *person (name (first, second, surname, title))* .
The relationship type *Purchase* is associated with the predicate variable

purchase(customer (person (name (first, second, surname, title)), ID),
store (name,address), typeOfPurchase, timeOfPurchase) .

The HERM predicate logic is inductively constructed in the same way as the predicate logic. Instead of simple variables we use structured variables. This language enables us to specify restriction on the scheme. For instance, the formula

part(barCode,color) $\wedge$ *part(barCode,color')* $\rightarrow$ *color = color'*

expresses that the same part with the same ID cannot have different color, i.e. parts can be identified by their bar code.

Queries can be expressed in a similar way. The query above is expressed by the logic program

q(x) $\leftarrow$ *purchase(customer(person(x),_), store("Safeway",_),_,_)* .

We can also specify behavior of a database over lifetime. A database is

modified by an action or more general by a transaction. Basic actions are queries or conditional manipulation operations. Manipulation operations such as *insert, delete, update* are defined in the HERM algebra. Database behavior can be specified on the basis of states. Given a HERM scheme $S = \{R_1, ...R_m\}$. A *state* is the set of classes $\{R_1^C, ...R_m^C\}$ with $R_i^C \in \mathcal{R}(R_i)$, $1 \leq i \leq m$ which satisfies certain restrictions Σ.

The structuring of the extended ER model allows to deduct a number of properties. As an example we consider the axiomatization of constraints generalizing those discussed in [SP84, Tha91]. We observe first that implication in the hierarchical predicate logic is reflexive, monotone, compact and closed. Let us consider classes of BV-constraints in HERM which form a cylindric algebra [Tsa89]. The order of constraints by $\Phi_{\models}$ possibly can be based on the order of of premises and conclusions. In this case the constraint set forms a pair algebra.

Observation 7.
Cylindric classes are pair algebras.

Examples of cylindric classes are the class of functional dependencies, the classes of Hungarian functional dependencies [Tha91], the class of inclusion dependencies and the class of multivalued dependencies. Further, the n-class of all $\geq n$-functional dependencies $X \rightarrow Y$ which left side contains at least n components and the class of rigid $\leq n$-inclusion dependencies $T_1[X] \subseteq T_2[X]$ which component list contain at most n components form a cylindric constraint set. Usually, union does not preserve cylindric sets.

Observation 8.
Cylindric constraint classes are axiomatized by reflexivity axioms, augmentation and transition rules.

If an axiomatization leads to reflexivity, augmentation and transitivity then union and decomposition rules can be deducted by the other rules. Transitivity may have to consider the specific influence of premises, e.g., transitivity for full multivalued dependencies is based on the root reduction rule [Tha91].

Based on this axiomatization we may introduce a general vertical decomposition form:
Given a schema structuring $\mathcal{S} = (\mathcal{ER}, \Sigma_{\mathcal{S}})$. A *vertical decomposition* of $\mathcal{S}$ is given a a mapping τ from $\mathcal{S}$ to $\mathcal{S}'$ which is defined by projection functions. The decomposition is *lossless* if a query q on $\mathcal{S}'$ can be defined such that for each db on $\mathcal{S}$ the equality $q(\tau(db)) = db$ is valid.
Let further Σ' the set of those constraints from $\Phi_{\models}(\Sigma)$ which are entirely defined on the structures in $\mathcal{S}'$. A decomposition based on projection is called *$\mathcal{C}$-constraint preserving* if $\Sigma \subseteq \Phi_{\models}(\Sigma')$.

Classical example of vertical decompositions are decompositions of relations to relations in the third normal form.

We may now introduce a general class of $\mathcal{C}$-decomposition algorithms:

Construct basic elements which are undecomposable.

Derive maximal elements by backward propagation of augmentation.

Reduce redundancy in the constraint set by backward propagation of transitivity.

Derive a left-right graph by associating conclusions of a constraint to the premise of another constraint.

Combine all minimal left sides of constraints which are not bound by another constraint to a group.

Derive projections based on all groups in the graph.

The first step of the decomposition algorithm is only introduced for convenience. This algorithm is a generalization of the classical synthesis algorithm.

Observation 9.
The $\mathcal{C}$-decomposition algorithm leads to $\mathcal{C}$-constraint preserving decomposition if the class $\mathcal{C}$ is cylindric.

3.2 Maturity and Capability of Object-Oriented / Relational Models

Object-oriented database models have been developed in order to overcome the impedance mismatch between languages for specification of structural aspects and languages for the specification of behavioral aspects. So far, no standard approach is known to object-orientation. *Objects* are handled in databases systems and specified on the basis of database models. They can own an *object identifier*, are structurally characterized by *values* and *references* to other objects and can posses their own *methods*, i.e.

$$o = (i, \{v\}, \{ref\}, \{meth\})$$

The value characterization is bound to a *structure* of the type T which is already defined. Characterizing properties of objects are described by *attributes* which form the structure of the object. Objects also have a *specific semantics* and a *general semantics*. The properties describe the *behavior* of objects. Objects which have the same structure, the same general semantics and the same operators are collected in *classes*. The structure, the operations and the semantics of a class are represented by *types* $T = (S, O, \Sigma)$. In this case, the modelling of objects includes the association of objects with classes C and their corresponding value type T and reference type R. Therefore, after classification the structure of objects is represented by

$$o = (i, \{(C, T, v)\}, \{(C, R, ref)\}, \{(T, meth)\}) \quad .$$

The recognized design methodologies vary in the scale of information modeled in the types. If objects in the classes can be distinguished by their values, then the identifiers can be omitted and we use *value-oriented modelling*. If this is not the case, we use an *object-oriented approach*. In the object-oriented approach, different approaches can be distinguished. If all objects are identifiable by their value types or by references to identifiable objects, then the database

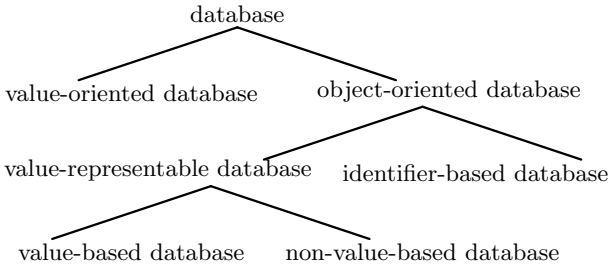


Figure 2: Classification of Database Models

is called *value-representable*. In this case, the database can also be modeled by the *value-oriented* approach, and a mapping from the value-representable scheme to a value-oriented scheme can be generated. If the database is not value-representable, then we have to use object identifiers. In this case either the identifier handling should be made public or else the databases cannot be updated and maintained. Therefore, value-representable databases are of special interest. Thus, we can distinguish database models as displayed in Figure 2.

It has been shown in [BT99, Sch94] that the concept of the object identifier can only be treated on the basis of higher-order epistemic and intuitionistic logics. Furthermore, identification by identifiers is different from identification by queries, equational logics and other identification methods. For this reason, the concept of the object identifier is far more complex than wanted and cannot be consistently and equivalently treated in database systems. Furthermore, methods can be generically derived from types only in the case if all objects are value-representable. Value-representable cyclic type systems require topos semantics[ST99] what is usually too complex to be handled in database systems. It can be shown that value-representable, non-cyclic type systems can be represented by value-oriented models.

3.3 XML - Couleur De Rose And Pitfalls

XML document specification and playout is based on a number of standards and co-standards:

XML documents are based on trees of elementary documents which are tagged by a begin and end delimiter.

Document schema specification is either based on

DTD specification which supports an entity-relationship modelling,

RDF schema reuse which allows to include other specifications, or

Schema specification which allows object-oriented modelling.

XLink, XPointer, XPath and XNamespace support a flexible parsing and playout enalbing documents to be extended by foreign sources which might be accessible.

XSL supports filtering, union and join of documents.

XML query languages add query facilities to XML document suites.

XML applications are supported by a large variety of application standards such as BizTalk and LOM.

This layering has a number of advantages and a number of disadvantages. The main advantage of this approach is that almost any object set can be represented by an XML document suite. XML documents may be well-formed. In this case they are semi-structured and may be represented by $\mathcal{A}$-trees which are defined by induction as follows

- each $\sigma \in \mathcal{A}$ is a tree, and
- if $\sigma \in \Sigma$ and $t_1, ..., t_n$ are trees then $\sigma(t_1, ..., t_n)$ is a tree.

The set $Dom(t) \subseteq \mathcal{N}^*$ of all nodes of a tree $t = \sigma(t_1, ..., t_n)$ is given by:
$Dom(t) = \{\epsilon\} \cup \{ui | i \in \{1, ..., n\}, u \in Dom(t_i)\}$
where ϵ is the root, ui is the ith child of u, and
$lab^t(u)$ is the label of u in t.

The disadvantages of XML steam from the generality of the approach. For instance, parsing of XML document sets must be supported by machines which are not less complex than Turing machines, i.e., tree automata
$M = (Q, \Sigma, \delta, (I_\sigma)_{\sigma \in \Sigma}, F), \quad F \subseteq Q, \delta : Q \times \Sigma \to 2^{Q^*}$.
A run $\lambda : Dom(t) \to Q$ specifies for each leave node $v : \epsilon \in \delta(\lambda(u1), lab^t(u))$ and for each node v with p children : $\lambda(u1)\lambda(u2)...\lambda(up) \in \delta(\lambda(u), lab^t(u))$.
The run accepts the tree t if $\lambda(\epsilon) \in F$.

XML document suites have, however, a number of other properties: they are partial and based on list semantics. Their implication is neither compact not monotone nor closed. Therefore, the axiomatization of XML constraints is more difficult compared with other database models. For instance, already the definition of keys and functional dependencies becomes a nightmare. The treatment of cardinality constraints is more difficult than for ER models. For instance, the definitions of [AFL02, BDF$^+$01] are incomplete since they do not consider the list-to-set axiom.

XML documents provide a universal structuring mechanism. [Kle07] has developed a modelling approach that limits the pitfalls of XML specification.

Finite implication of path constraints is co-r.e. complete and implication is r.e. complete for semi-structured models. The implication problem for key constraints is harder than in the relational and ER case. It involves implication on regular path expressions which is known to be PSPACE-hard. The satisfiability problem for key constraints and referential inclusion constraints becomes

undecidable in the presence of DTD's. For this reason, simpler language must be used for specification of constraints in XML.

4 Mappings That Preserve Semantic Invariants

4.1 Mappings From Entity-Relationship Structures To Relations

ER models have a rich structuring and a rich set of integrity constraints. Relational DBMS usually do not support such rich structuring facilities. Therefore, mappings of richly structured HERM schemata to relational technology is forgetful in the sense that structures or integrity constraints cannot be represented within the structuring provided by relational DBMS. We can, however, represent static integrity constraints by transition constraints. In this case, we do not loose integrity constraints which primary role is to be a filter of invalid data. The entire translation strategy is discussed in [Tha00].

4.2 From Entity-Relationship Databases To XML Suites

DTD's and XSchema do not allow a semantics preserving translation. So we might either develop a normalization approach such that all constraints are representable within the DTD's or XSchema language or we use XML only for playout of content and for exchange of content between views or views and the database. We use the second approach and maintain consistency through the database system. Therefore, we only need a translation procedure for HERM views to XML suite schemes. The layering approach to the generation of XML allows to use another strategy to generate XML documents. This facility is displayed in Figure 3.

This transformation approach is already used in the DaMiT project and implements an XML suite on top of the relational DBMS DB2. The extended ER model [Tha00] provides a better approach to XML suite generation than relational models or the classical ER model for a number of reasons:

- Structures can be defined already in complex nested formats.
- Types of higher order are supported.
- The model uses cardinality constraints with participation semantics.

We claim that the extended ER model is better suited than UML since:

- Types have a clearly defined semantics.
- Schema components are integrated.
- HERM has a development methodology that supports consistency and completeness and thus leads to schemata of high quality.

We observe further that well-structured XML may be considered to be a restricted form of HERM:

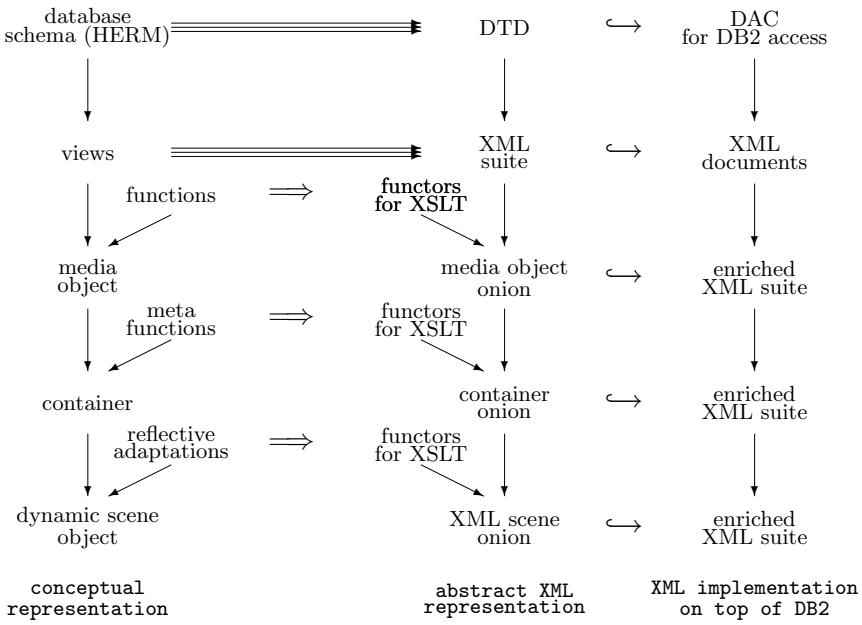


Figure 3: The General Procedure for Translation from SiteLang to XML

- XML Schema and XForms are suited for defining hierarchical extracts of HERM.
- HERM specialization is based on type specialization.
- Unary cardinality constraints are supported. If more complex constraints are required we may use vertical decomposition approaches.
- Variants of web objects may be referenced by an annotated XDNL approach.

Therefore, suites of restricted XML documents may be understood as object-oriented hierarchical database. If documents are reused by other documents we associate them via XDNL variants.

Translation of HERM schemata to DTD can be based on a number of approaches which are similar to the translation approaches used for transformation of schemata to hierarchical database schemata:

Full type separation: Each entity, relationship and cluster type is represented by their own `<!ELEMENT ..>` representation. All entity types have an `ID` which is used through `IDREF` by other types.

Small star schema representation: The central type of a star schema is represented by its own `<!ELEMENT ..>` representation which uses components for association to other central types. Star associations to other types are represented through attribute lists and the `IDREF #REQUIRED` data type.

We observe that the translation is not semantics preserving. All referential constraints must be maintained through application programs.

The translation from HERM schemata to XML suites is based on the following steps which are similar to those in [Tha00] for transformation of HERM schemata to hierarchical database structures:

- HERM translation to hierarchical database structures is based on the introduction of copies in the master-slave mode. Master objects have their ID. Slave objects do not have own data. They are only mirrors of the master objects. The translation procedure for XML transformation is based on two assumptions:
  - Strong aggregations are exclusive, i.e., a components belongs to one and only one supertype.
  - Weak non-exclusive aggregations are mapped to mirrored types.

  The translation may consider interaction on the basis of the story space.
- The transformation process is a stepwise transformation starting with HERM types of order 0, continuing with types of order (i+1) after types of order i have been translated.

  - Attribute types are directly transformed to corresponding DTD structures. Attributes are exclusively used by their entity types and thus form an attribute structure of the DTD structures.
  - Entity types are directly translated to DTD structures and use the attribute type translations.
  - Relationship types are transformed to DTD types. They can be integrated if cardinality constraints are (1,1)-(1,1) constraints.
  - Hierarchies use the based-on construction which is similar to the one of DTD schemata.
  - Cluster types use the mirroring approach for their transformation.

- Views are queries and thus can be transformed to XML queries. We distinguish between retrieval views and modification views. The first are used for the generation of media objects. The later are used for modification of the database.

- Integrity constraints are handled similarly to the approaches in relational databases. Since XML suites are generated on top of databases integrity is maintained inside the database system.

Since we use XML suites on top of database systems we are able to handle consistency of XML suites without inheriting the difficulties of XML document sets.

5 Concluding

Semantical and relational databases are coherently specified sets. They are based on rigid structuring. The advantage of such rigidity is the derivability of functionality for querying, updating, transaction, concurrency, integrity ect. A large set of integrity constraints have been developed and supported in the database setting. XML is far less rigid. The flexibility of XML has a price which might be too high: the loss of a general definition frame for structuring and especially integrity constraints. XML suites are not supportable in the same form. Already simple classes of constraints such as key constraints are challenging. Simple constraint sets such as key constraints and referential integrity constraints are very difficult to handle. There are two approaches to master this situation: either to build sophisticated engines or to restrict XML suites. We advocate the second approach: *XML suites as playout and exchange documents on top of database technology*.

References

[AFL02] M. Arenas, W. Fan, and L. Libkin. On veriyfying consistency of XML specifications. In *Proc. ACM PODS*, pages 259–270. ACM, 2002.

[BDF+01] P. Buneman, S. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Keys for XML. In *Proc. WWW 10*, pages 201–210. ACM, 2001.

[BT99] C. Beeri and B. Thalheim. Identification as a primitive of database models. In *Proc. FoMLaDO'98*, pages 19–36. Kluwer, London, 1999.

[GP83] M. Gyssens and J. Paredaens. Another view of functional and multivalued dependencies in the relational database model. *Int. J. Computer and Information Sciences*, 12:247–267, 1983.

[GP86] M. Gyssens and J. Paredaens. On the decomposition of join dependencies. *Advances in Computing Research*, 3:69–106, 1986.

[GPG88] M. Gyssens, J. Paredaens, and D. Van Gucht. A uniform approach towards handling atomic and structural information in the nested relational database model. Technical Report UIA 88-17, University of Antwerp, 1988.

[HPT88] G. J. Houben, J. Paredaens, and D. Tahon. Expressing structural information by the nested relational algebra: An overview. In *Computing Science Notes, Proc. 8th Int. Conf. on Computer Science - SCCC'88*, Santiago de Chile, 1988.

[Kle07] M. Klettke. *Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen*. Advanced PhD (Habilitation Thesis), Rostock University, Faculty for Computer Science and Electronics, 2007.

[Lin03] S. Link. Consistency enforcement in databases. In *Proc. Semantics in Databases*, volume 2582 of *LNCS*, pages 139–159. Springer, Heidelberg, 2003.

[Par82] J. Paredaens. A universal formalism to express decompositions, functional dependencies and other constraints in a relational data base. *TCS*, 19(2):143–163, 1982.

[PBGG89] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The structure of the relational database model*. Springer, Berlin, 1989.

[Sch94] K.-D. Schewe. *The specification of data-intensive application systems*. Advanced PhD (Habilitation Thesis), Brandenburg University of Technology at Cottbus, Faculty of Mathematics, Natural Sciences and Computer Science, 1994.

[SP84] A. A. Stognij and W. W. Pasitschnik. Relational models of databases. Technical report, Institut Kibernetiki, Kiew, 1984. In Russian.

[ST99] K.-D. Schewe and B. Thalheim. A generalization of Dijkstra's calculus to typed program specifications. In *Proc. FCT '99*, volume 1684 of *LNCS*, pages 463–474. Springer, 1999.

[SW05] G. Simsion and G.C. Witt. *Data modeling essentials*. Morgan Kaufmann, San Francisco, 2005.

[Tha91] B. Thalheim. *Dependencies in relational databases*. Teubner, Leipzig, 1991.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000.

[Tsa89] M. Sh. Tsalenko. *Modeling of semantics for databases*. Nauka, Moscov, 1989. In Russian.