Dietmar Seipel
José María Turull-T

Foundation of Informa Knowledg

LNCS 2942

Third International Sympos
Wilhelminenburg Castle, A
Proceedings

Lecture Notes in Computer Science 2942

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

oarieli@mta.ac.il

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

oarieli@mta.ac.il

Dietmar Seipel José María Turull-Torres (Eds.)

Foundations of Information and Knowledge Systems

Third International Symposium, FoIKS 2004
Wilheminenburg Castle, Austria, February 17-20, 2004
Proceedings

Springer

oarieli@mta.ac.il

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Dietmar Seipel
University of Würzburg
Department of Computer Science
Am Hubland, 97074 Würzburg, Germany
E-mail: seipel@informatik.uni-wuerzburg.de

José María Turull-Torres
Massey University
Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
E-mail: J.M.Turull@massey.ac.nz

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.2, H.3, H.5, I.2.4, F.3.2, G.2

ISSN 0302-9743
ISBN 3-540-20965-4 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10983706 06/3142 5 4 3 2 1 0

oarieli@mta.ac.il

Preface

This volume contains the papers presented at the 3rd International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2004), which was held in Castle Wilhelminenberg, Vienna, Austria, from February 17th to 20th, 2004.

FoIKS is a biennial event focussing on theoretical foundations of information and knowledge systems. It aims at bringing together researchers working on the theoretical foundations of information and knowledge systems and attracting researchers working in mathematical fields such as discrete mathematics, combinatorics, logics, and finite model theory who are interested in applying their theories to research on database and knowledge base theory.

FoIKS took up the tradition of the conference series Mathematical Fundamentals of Database Systems (MFDBS) which enabled East–West collaboration in the field of database theory. The first FoIKS symposium was held in Burg, Spreewald (Germany) in 2000, and the second FoIKS symposium was held in Salzau Castle (Germany) in 2002. Former MFDBS conferences were held in Dresden (Germany) in 1987, Visegrád (Hungary) in 1989, and in Rostock (Germany) in 1991. Proceedings of these previous events were published by Springer-Verlag as volumes 305, 364, 495, 1762, and 2284 of the LNCS series, respectively.

In addition the FoIKS symposium was intended to be a forum for intensive discussions. For this reason the time slots for long and short contributions were 50 and 30 minutes, respectively, followed by 20 and 10 minutes for discussions, respectively. Furthermore, participants were asked in advance to prepare to act as correspondents for the contributions of other authors. There were also special sessions for the presentation and discussion of open research problems.

The FoIKS 2004 call for papers solicited contributions dealing with any foundational aspect of information and knowledge systems, e.g.,

- mathematical foundations: discrete methods, boolean functions, finite model theory
- database design: formal models, dependency theory, schema translations, desirable properties
- query languages: expressiveness, computational and descriptive complexity, query languages for advanced data models, classifications of computable queries
- semi-structured databases and WWW: models of Web databases, querying semi-structured databases, Web transactions and negotiations
- security in data and knowledge bases: cryptography, steganography, information hiding

oarieli@mta.ac.il

- integrity and constraint management: verification, validation, and enforcement of consistency, triggers
- information integration: heterogenous data, views, schema dominance and equivalence
- database and knowledge base dynamics: models of transactions, models of interaction, updates, consistency preservation, concurrency control
- intelligent agents: multi-agent systems, autonomous agents, foundations of software agents, cooperative agents
- logics in databases and AI: non-classical logics, spatial and temporal logics, probabilistic logics, deontic logic, logic programming
- knowledge representation: planning, description logics, knowledge and belief, belief revision and update, non-monotonic formalisms, uncertainty
- reasoning techniques: theorem proving, abduction, induction, constraint satisfaction, common-sense reasoning, probabilistic reasoning, reasoning about actions.

The programme committee received 64 submissions. Each paper was carefully reviewed by at least two experienced referees, and most of the papers were reviewed by three referees. Fourteen papers were chosen for long presentations and four papers for short presentations. This volume contains versions of these papers polished based on the comments made in the reviews. A few papers will be selected for further extension and publishing in a special issue of the journal Annals of Mathematics and Artificial Intelligence.

We would like to thank all authors who submitted papers and all workshop participants for the fruitful discussions. We are grateful to the members of the programme committee and the external referees for their timely expertise in carefully reviewing the papers, and we would like to express our thanks to our hosts for the beautiful week in the pleasant surroundings of Castle Wilhelminenberg near Vienna.

February 2004

Dietmar Seipel
José María Turull-Torres

oarieli@mta.ac.il

Programme Committee Co-chairs

Dietmar Seipel	University of Würzburg (Germany)
José María Turull-Torres	Massey University (New Zealand)

Programme Committee

Leopoldo Bertossi	Carleton University, Ottawa (Canada)
Joachim Biskup	University of Dortmund (Germany)
François Bry	University of Munich (Germany)
Samir Chopra	University of New South Wales (Australia)
Jürgen Dix	University of Manchester (UK)
Thomas Eiter	Vienna University of Technology (Austria)
Jörg Flum	University of Freiburg (Germany)
Burkhard Freitag	University of Passau (Germany)
Stephen J. Hegner	Umeå University (Sweden)
Lauri Hella	University of Tampere (Finland)
Sven Hartmann	Massey University (New Zealand)
Gyula Katona	Hungarian Academy of Sciences (Hungary)
Gabriele Kern-Isberner	FernUniversität Hagen (Germany)
Hans-Joachim Klein	University of Kiel (Germany)
Alberto Mendelzon	University of Toronto (Canada)
Jack Minker	University of Maryland (USA)
Kotagiri Ramamohanarao	University of Melbourne (Australia)
Vladimir Sazonov	University of Liverpool (UK)
Klaus-Dieter Schewe	Massey University (New Zealand)
Thomas Schwentick	Philipps University at Marburg (Germany)
Dietmar Seipel	University of Würzburg (Germany)
Bernhard Thalheim	University of Kiel (Germany)
José María Turull-Torres	Massey University (New Zealand)

oarieli@mta.ac.il

External Referees

Matthias Beck	Christoph Beierle
Gerhard Bloch	Gerd Brewka
Yijia Chen	Marina de Vos
Janos Demetrovics	Scott Dexter
Claus Dziarstek	Heinz-Dieter Ebbinghaus
Klaus Fellbaum	Flavio A. Ferrarotti
Hans-Peter Gumm	Michael Guppenberger
Wolfgang Hesse	Marbod Hopfner
Eyke Hüllermeier	Roland Kaschek
Kathrin Konczak	Boris Konev
Georg Lausen	Jens Lechtenbörger
Sebastian Link	Alexei Lisitsa
Thomas Meyer	Michael J. Minock
Bernhard Nebel	Thomas Nitsche
Maurice Pagnucco	Jari Palomaki
Gerald Pfeifer	Flavio Rizzolo
Attila Sali	Torsten Schlieder
Petra Schwaiger	Alexei Tretiakov
Bela Uhrin	Alejandro A. Vaisman
Wiebe van der Hoek	Mark Weyer

Organization

Thomas Eiter	Vienna University of Technology (Austria)

oarieli@mta.ac.il

Table of Contents

Invited Talks

Regular Papers

oarieli@mta.ac.il

oarieli@mta.ac.il

Hypergraph Transversals

Georg Gottlob

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9–11, A-1040 Wien, Austria
gottlob@dbai.tuwien.ac.at

Hypergraph Transversals have been studied in Mathematics for a long time, cf. [2]. Generating minimal transversals of a hypergraph is an important problem which has many applications in Computer Science, especially in database Theory, Logic, and AI. We briefly survey some results on problems which are known to be related to computing the transversal hypergraph, where we focus on problems in database theory, propositional Logic and AI (for a more detailed survey and further references cf. [10]).

A *hypergraph* $\mathcal{H} = (V, E)$ consists of a finite collection E of sets over a finite set V. The elements of E are called *hyperedges*, or simply *edges*. A *transversal* (or *hitting set*) of $\mathcal{H}$ is a set $T \subseteq V$ that meets every edge of E. A transversal is *minimal*, if it does not contain any other transversal as a subset. The set T of all minimal transversals of $\mathcal{H} = (V, E)$, constitutes together with V also a hypergraph $Tr(\mathcal{H}) = (V, T)$, which is called the *transversal hypergraph* of $\mathcal{H}$. The famous Transversal Hypergraph Problem (TRANS-HYP) is then as follows: Given two hypergraphs $\mathcal{G} = (V, E)$ and $\mathcal{H} = (V, F)$ on a finite set V, decide whether $\mathcal{G} = Tr(\mathcal{H})$ holds. The corresponding computation problem is called Transversal Enumeration (TRANS-ENUM) and is phrased as follows: Given a hypergraph $\mathcal{H} = (V, E)$ on a finite set V, compute the transversal hypergraph $Tr(\mathcal{H})$.

From the point of computability in polynomial time, the decisional and the computational variant of the transversal hypergraph problem are in fact equivalent: It is known that, for any class $\mathcal{C}$ of hypergraphs, TRANS-ENUM is solvable in *polynomial total time* (or *output-polynomial time*), i.e., in time polynomial in the combined size of $\mathcal{H}$ and $Tr(\mathcal{H})$, if and only if TRANS-HYP is in the class P for all pairs $(\mathcal{H}, \mathcal{G})$ such that $\mathcal{H} \in \mathcal{C}$ [3].

The precise complexity of TRANS-HYP is not known to date, and is in fact open for more than 20 years now. Accordingly, it is unknown whether TRANS-ENUM can be solved in output-polynomial time.

The problems TRANS-HYP and TRANS-ENUM have a large number of applications in many areas of Computer Science, including Distributed Systems, Databases, Boolean Circuits and Artificial Intelligence. There, they have important applications in Diagnosis, Machine Learning, Data Mining, and Explanation Finding, see e.g. [9,12,17,20,21,22,24] and the references therein.

We call a decision problem Π TRANS-HYP-*hard*, if problem TRANS-HYP can be reduced to it by a standard polynomial time transformation. Furthermore, Π is TRANS-HYP-*complete*, if Π is TRANS-HYP-hard and, moreover, Π can be polynomially transformed into TRANS-HYP; that is, Π and TRANS-HYP are

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 1–5, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

equivalent modulo polynomial time transformations. We use analogous terminology of TRANS-ENUM-*hardness* and TRANS-ENUM-*completeness* for computations problems, i.e., problems with output (for more details, cf. [10]).

Let us first discuss issues of *structural* complexity. In a landmark paper, Fredman and Khachiyan [15] proved that TRANS-HYP can be solved in time $n^{o(\log n)}$, and thus in quasi-polynomial time. This shows that the problem is most likely not co-NP-complete, since no co-NP-complete problem is known which is solvable in quasi-polynomial time; if any such problem exists, then all problems in NP and co-NP can be solved in quasi-polynomial time.

A natural question is whether TRANS-HYP lies in some lower complexity class based on other resources than just runtime. In a recent paper [11], it was shown that the complement of this problem is solvable in polynomial time with *limited nondeterminism*, i.e, by a nondeterministic polynomial-time algorithm that makes only a poly-logarithmic number of guesses in the size of the input. For a survey on complexity classes with limited nondeterminism, and for several references see [18]. More precisely, [11] shows that non-duality of a pair $\mathcal{G}, \mathcal{H})$ can be proved in polynomial time with $O(\chi(n) \cdot \log n)$ suitably guessed bits, where $\chi(n)$ is given by $\chi(n)^{\chi(n)} = n$; note that $\chi(n) = o(\log n)$.

This result is surprising, because most researchers dealing with the complexity of the transversal hypergraph thought so far that these problems are completely unrelated to limited nondeterminism.

A large number of tractable restrictions of TRANS-HYP and TRANS-ENUM are known in the literature, e.g. [7,5,4,8,9,13,11,16,23,27], and references therein.

Examples of tractable classes are instances $(\mathcal{H}, \mathcal{G})$ where $\mathcal{H}$ has the edge sizes bounded by a constant, or where $\mathcal{H}$ is *acyclic*. Various "degrees" of hypergraph acyclicity have been defined in the literature [14]. The most general notion of hypergraph acyclicity (applying to the largest class of hypergraphs) is α-acyclicity; less general notions are (in descending order of generality) β-, γ-, and Berge-acyclicity (see [14]). In [9], it was shown that Hypergraph transversal instances with β-acyclic $\mathcal{H}$ are tractable. In [11], this tractability result has been recently improved to instances where $\mathcal{H}$ is α-acyclic and simple. This result is a corollary to a more general tractability result for hypergraphs whose *degeneracy* is bounded by a constant; simple, α-acyclic hypergraphs have degeneracy 1.

Furthermore, [11] shows that instances $(\mathcal{H}, \mathcal{G})$ of TRANS-HYP where the vertex-hyperedge incidence graphs of $\mathcal{H}$ (or of $\mathcal{G}$) have *bounded treewidth* are solvable in polynomial time.

In the sequel, we mention a few problems closely related to TRANS-HYP or TRANS-ENUM. These and many other such problems are discussed in detail in [10,9].

Functional Dependency Inference

Given a database relation instance r and a set of functional dependencies (FDs) F, deciding whether F characterizes precisely the FDs holding on r, i.e., deciding whether F^+ is identical with the set $FD(r)$ of all FDs valid on r, is TRANS-HYP-hard and in co-NP. The precise complexity is currently open. The same problem

oarieli@mta.ac.il

becomes TRANS-HYP-complete if F is in Boyce-Codd Normal Form (BCNF), i.e., iff all left hand sides of the FDs in F are keys. Given a relation instance r, generating a cover F of FDs suchthat $F^+ = FD(r)$ is TRANS-ENUM-hard. Vice-versa, given a set F, generating a so-called Armstrong relation, i.e., a relation instance r incorporating precisely all FDs of F^+ is also TRANS-ENUM hard. For more details on the Dependency Inference problem and on related problems cf. [24,25,19,26,10], as well as the extended version of [9].

Data Mining: Maximal Frequent Sets

Given a 0/1 $m \times n$ matrix A and an integral threshold t, associate with each subset $C \subseteq \{1, \ldots, n\}$ of column indices the subset $R(C)$ of all rows $r \in \{1, \ldots, m\}$ in A such that $A(r, j) = 1$ for every $j \in C$. Then C is called *frequent*, if $|R(C)| \geq t$, and C is called *infrequent*, if $|R(C)| < t$. Let us denote by $F_t(A)$ and $\hat{F}_t(A)$ the sets of all frequent and infrequent column sets C in A, respectively.

The generation of frequent and infrequent sets in A is a key problem in knowledge discovery and data mining, which occurs in mining association rules, correlations, and other tasks. Of particular interest are the maximal frequent sets $M_t \subseteq F_t$ and the minimal infrequent sets $I_t \subseteq \hat{F}_t$, since they mark the boundary of frequent sets (both maximal and minimal under set inclusion). The following result has been recently proved in [6]: *The problem of computing, given a 0/1 matrix A and a threshold t, the sets M_t and I_t is* TRANS-ENUM-*complete.*

Theory Approximation: Horn Envelope

A logical theory Σ is *Horn*, if it is a set of Horn clauses, i.e., disjunctions $l_1 \vee \cdots \vee l_m$ of literals l_i such that at most one of them is positive. Semantically, Horn theories are characterized by the property that their set of models, $mod(\Sigma)$, is closed under intersection, i.e., $M, M' \in mod(\Sigma)$ implies $M \cap M' \in mod(\Sigma)$; here, $M \cap M'$ is the model M'' which results by atomwise logical conjunction of M and M', i.e., $M'' \models a$ iff $M \models a$ and $M' \models a$, for every atom a.

Any theory Σ has a unique *Horn envelope*, which is the strongest (w.r.t. implication) Horn theory Σ' such that $\Sigma \models \Sigma'$. The Horn envelope might be represented by different Horn theories, but there is a unique representation, which we denote by $HEnv(\Sigma)$, which consists of all prime clauses of Σ'. The following result was established in [22], where the TRANS-ENUM hardness part was proved in [21]: *The problem of computing, given the models $mod(\Sigma)$ of a propositional theory Σ, the Horn envelope $HEnv(\Sigma)$ is* TRANS-ENUM-*complete.*

Dualization of Boolean Functions

There is a well-known and close connection of TRANS-ENUM to the well-known dualization problem of Boolean Functions: given a CNF ϕ of a Boolean function f, compute a prime CNF ψ of its dual f^d, i.e., the function which has value 1 in input vector $b = (b_1, \ldots, b_n)$ iff f has value 0 on the input vector $\overline{b} =$

oarieli@mta.ac.il

$(b_1 \oplus 1, \ldots, b_n \oplus 1)$. Special algorithms for the dualization problem can be tracked down at least to the 60's of the past century, cf. [1]. It is not hard to see that this problem is intractable; in fact, its decisional varian DUAL which given two CNFs φ and ψ of Boolean functions f and g, respectively. consists in deciding whether φ and ψ represent a pair (f, g) of dual Boolean functions is co-NP-complete, where hardness holds even if ψ is asserted to be a prime CNF of g.

In case of monotone Boolean functions, the dualization problem is equivalent to determining the transversal hypergraph. Let MONOTONE DUALIZATION designate the subcase of DUALIZATION where f is a monotone Boolean function, and similarly MONOTONE DUAL the subcase of DUAL where f is a monotone Boolean function.

The following statement summarizes well-known results that are part of the folklore: MONOTONE DUALIZATION *is* TRANS-ENUM-*complete, and* MONOTONE DUAL *is* TRANS-HYP-*complete.* For a proof see [10].

Restricted Versions of the Satisfiability Problem

We denote by IMSAT (Intersecting Monotone SAT) the restriucted version of the classical SAT problem where each clause contains either only positive literals or only negative literals, and where every pair of a positive and a negative clause resolves, i.e., there is at least one atom which occurs unnegated in the positive clause and negated in the negative claues. The following holds [9,10]: IMSAT *is co-*TRANS-HYP-*complete.* This complexity result holds even if we restrict IMSAT instances to clause sets $\mathcal{C}$ where the negative clauses are precisely all clauses C^- such that $C^- = \{\neg u : u \in C^+\}$ for some positive clause $C^+ \in \mathcal{C}$.

References

1. C. Benzaken. Algorithme de dualisation d'une fonction booléenne. *Revue Francaise de Traitment de l'Information – Chiffres*, 9(2):119–128, 1966.
2. C. Berge. Hypergraphs. North Holland, 1989.
3. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123:50–63, 1995.
4. E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Processing Letters*, 10(4):253–266, 2000.
5. E. Boros, V. Gurvich, and P. L. Hammer. Dual subimplicants of positive Boolean functions. *Optimization Methods and Software*, 10:147–156, 1998.
6. E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In *Proc. STACS-02*, LNCS 2285, pp. 133–141, 2002.
7. E. Boros, P. Hammer, T. Ibaraki, and K. Kawakami. Polynomial time recognition of 2-monotonic positive Boolean functions given by an oracle. *SIAM J. Comput.*, 26(1):93–109, 1997.
8. C. Domingo, N. Mishra, and L. Pitt. Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. *Machine Learning* 37, 1999.

oarieli@mta.ac.il

9. T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
10. T. Eiter and G. Gottlob. Hypergraph Transversal Computation and Related Problems in Logic and AI *Proceedings 8th European Conference on Logics in Artificial Intelligence (JELIA 2002)*, Springer LNCS 2424, pp. 549–564, 2002.
11. T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003. Preprint available as *Computer Science Repository Report (CoRR) nr. cs.DS/0204009* via URL: `http://arxiv.org/abs/cs/0204009`. (A shorter version has appeared in *Proc. ACM STOC-2002*, pp. 14–22, 2002.)
12. T. Eiter and K. Makino. On computing all abductive explanations. In *Proc. 18th National Conference on Artificial Intelligence (AAAI '02)*. AAAI Press, 2002.
13. T. Eiter, K. Makino, and T. Ibaraki. Decision lists and related Boolean functions. *Theoretical Computer Science*, 270(1–2):493–524, 2002.
14. R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.
15. M. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.
16. D. Gaur and R. Krishnamurti. Self-duality of bounded monotone Boolean functions and related problems. In *Proc. 11th International Conference on Algorithmic Learning Theory (ALT)*, LNCS 1968, pp. 209–223. Springer, 2000.
17. G. Gogic, C. Papadimitriou, and M. Sideri. Incremental recompilation of knowledge. *J. Artificial Intelligence Research*, 8:23–37, 1998.
18. J. Goldsmith, M. Levy, and M. Mundhenk. Limited nondeterminism. *SIGACT News*, 27(2):20–29, 1978.
19. G. Gottlob and L. Libkin. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica*, 9(4):385–402, 1990.
20. D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proc. 16th ACM Symp. on Principles of Database Systems (PODS-97)*, pp. 209–216, 1997.
21. D. Kavvadias, C. Papadimitriou, and M. Sideri. On Horn envelopes and hypergraph transversals. In W. Ng, editor, *Proc. 4th International Symposium on Algorithms and Computation (ISAAC-93)*, LNCS 762, pp. 399–405, 1993.
22. R. Khardon. Translating between Horn representations and their characteristic models. *J. Artificial Intelligence Research*, 3:349–372, 1995.
23. K. Makino and T. Ibaraki. A fast and simple algorithm for identifying 2-monotonic positive Boolean functions. *Journal of Algorithms*, 26:291–302, 1998.
24. H. Mannila and K.-J. Räihä. Design by Example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 22(2):126–141, 1986.
25. H. Mannila and K.-J. Räihä. Dependency Inference. *VLDB 1987*, pp.155–158.
26. H. Mannila and K.-J. Räihä. On the Complexity of Inferring Functional Dependencies. *Discrete Applied Mathematics*, 40(2):237–23, 1992.
27. N. Mishra and L. Pitt. Generating all maximal independent sets of bounded-degree hypergraphs. In *Proc. Tenth Annual Conference on Computational Learning Theory (COLT-97)*, pp. 211–217, 1997.

oarieli@mta.ac.il

Abstract State Machines: An Overview of the Project

Yuri Gurevich

Microsoft Research, One Microsoft Way, Redmond, WA 98052

Abstract. This is an extended abstract of an invited talk at the Third International Symposium on Foundations of Information and Knowledge Systems to be held in Vienna, Austria, in February 2004.
We quickly survey the ASM project, from its foundational roots to industrial applications.

1 Prelude

The ASM project started as a foundational investigation by this theorist in the mid 1980s at the University of Michigan. In the 1990s, a community of ASMers formed, see the ASM academic website [1], and several engines to write and execute abstract state machines were developed. Siemens was the first large company to use such engines. In 1998, Microsoft Research invited this theorist to build a group on Foundations of Software Engineering (FSE) and to apply the ASM theory. We sketch these developments stressing the foundational issues.

A draft of the talk, in the form of Power-Point slides, was written first; hence the choppy character of this hastily written extended abstract.

Many thanks to Andreas Blass and Jon Jacky for useful comments.

2 The Original Foundational Problem

What is computer science about mathematically speaking? To elucidate the question, let's compare computer science to physics. The physicists use partial differential equations (PDEs) to model the physical world. What mathematics should play the role of PDEs in computer science?

The world of computer science is much different from that of physics. The precise state of a physical system is an abstraction, but the state of a computer (as a digital rather than physical system) is examinable. Much of physics is devoted to continuous processes where the very next state of the process does not exist. Computer scientists are interested primarily in discrete processes where the next state is well defined (unless the process stops). Here we concentrate on discrete processes exclusively.

Computer science isn't even a natural science: we study the artificial world of computers. Imagine intelligent visitors from a distant planet. Their mathematics and physics are probably similar to ours, but their computers and therefore their

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 6–13, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

computer science may be vastly different. And yet there is something objective (and going beyond the classical analysis of which functions are computable) about computations that is worth exploring.

2.1 Dynamics

The classical mathematical approach to deal with dynamics is to reduce it to statics. Instead of analyzing motion directly, mathematicians analyze the history of motion. Your (for simplicity, ordinary) differential equations may speak about $dx/dt, dy/dt, \ldots$ where t is time, and so you seemingly study a process developing in time. But a simple coordinate transformation allows you to rewrite your equations in terms of $dt/dx, dy/dx, \ldots$, and now the process develops along a spatial line rather than the time line. This shows that time is just another dimension in the perfectly static history of the process. In that sense your (possibly most fruitful) analysis is sort of an autopsy.

This reduction of dynamics to statics does not come for free. We illustrate this on the case of a program with three integer variables x_1, x_2, x_3 that goes from the initial state X_0 to states $X_1, X_2, \ldots$. If we make the (logical) time explicit (so that the state at time t is X_t), then every x_i is a function $x_i(t)$ of t. For simplicity, we assume that every state X_t is uniquely defined by the values $x_1(t), x_2(t), x_3(t)$. Every $x_i(t+1)$ depends on X_t and therefore on $x_1(t), x_2(t), x_3(t)$. Even if the original program was simple, the derived system of equations may be hard.

The ingenious mathematical analysis fakes a discretization of a continuous process and concentrates on the relation between the current state (at time t) and the "next state" (at time $t+dt$). We don't have to fake discretization because we deal with discrete processes to begin with.

Let's start our analysis of algorithms with algorithms that work in sequential time. The term algorithm is understood broadly; every computer system at any fixed abstraction level is an algorithm. A sequential time algorithm starts in some state X_0 and proceeds to states $X_1, X_2, \ldots$.

Postulate 1 (Sequential Time [10]) *The behavior of a sequential time algorithm is determined by the set of states, the subset of initial states, and the state transition function.*

Remark 1 Shouldn't it be the state transition relation rather than the state transition function? By default, an algorithm is deterministic. (One can argue that algorithms are intrinsically deterministic; see [10, Section 9] in this connection.) But nondeterministic algorithms (and nondeterministic abstract state machines) make appearance in the sequel.

A finite state machine is an example of sequential time algorithm. In general, a sequential time algorithm is a finite state machine or an infinite state machine. The computation theory offers us the universal Turing machine [13]. But it is clearly inadequate to describe arbitrary sequential time algorithms succinctly. Can one improve on Turing's machine?

oarieli@mta.ac.il

2.2 Statics

It occurred to us early on that every static mathematic reality can be described as a structure in the sense of mathematical logic, that is a set with operations and relations. Next time you talk to a mathematician, ask him what he is working on. Whether he/she works with graphs or Banach spaces or whatever, it surely will be some kind of structures. This insight eventually gave rise to the Abstract State Postulate. We give here only an abbreviated version of the postulate.

Postulate 2 (Abstract State [10]) *The states of an algorithm are structures of a fixed vocabulary. . . .*

The vocabulary is intrinsic to the algorithm itself and does not depend on the input or state. The current state contains all other information that the algorithm needs to continue the computation.

Remark 2 We use the notion of structure in a slightly unorthodox way. We presume that the base set of every structure contains the ideal elements `true` and `false` and that predicates are operations taking value in {`true`,`false`}. It follows that our states are algebras in the sense of the science of universal algebra.

3 Abstract State Machines

What is the true state of a program in, say, the C programming language? Often, they tell you that the state is given by the values of its variables. This is not true. You need to know also the procedure stack and where the program counter is.

The Key Observation. With fully transparent states (defined exclusively by the values of the variables), a simple programming language suffices to program transitions.

Note that the state of a C program can be made fully transparent by means of auxiliary variables [11]. The same applies to every other programming language.

The key observation led to the definition of abstract state machines, or ASMs.

Remark 3 We view a computation as an evolution of the state. According to the abstract state postulate and Remark 1, states are algebras. Hence the original name "evolving algebras" for abstract state machines.

We consider three categories of algorithms: sequential, (synchronous) parallel, and distributed. The definition of sequential ASMs was formulated in [8]. The definitions of parallel ASMs and distributed ASMs were formulated in [9]. Numerous examples of ASMs are found on the academic ASM site [1]. In the talk we illustrate ASM definitions by means of examples.

oarieli@mta.ac.il

4 The Foundational Ambition of the ASM Project

The ASM Thesis. *Every algorithm is an ASM as far as the behavior is concerned. In particular the given algorithm can be step-for-step simulated by an appropriate ASM.*

This bold (impudent?) thesis was formulated in [9]. Recall that the notion of algorithm is understood broadly: every computer system at a fixed level of abstraction is an algorithm.

4.1 Theoretical Confirmation of the ASM Thesis

Intuitively, a sequential algorithm is a sequential time algorithm with steps of bounded complexity. In the presence of the sequential time postulate and the abstract state postulate, an additional *Bounded Exploration Postulate* expresses that the steps of any sequential algorithm have bounded complexity.

In [10], a sequential algorithm is defined as anything that satisfies these three postulates: the sequential time postulate, the abstract state postulate, and the bounded exploration postulate. Sequential ASMs are sequential algorithms of course. Two sequential algorithm are *behaviorally identical* if they have the same states, the same initial states and the same state transition function.

Theorem 1 (Sequential Characterization Theorem [10]) *For every sequential algorithm, there is a behaviorally identical sequential ASM.*

In [4], a parallel algorithm is defined as anything satisfying the sequential time postulate, the abstract state postulate, and several other postulates describing how the parallel subprocesses communicate with each other. The definition of parallel ASMs in [4] is a variant of that in [9]. In either version, parallel ASMs are parallel algorithms.

Theorem 2 (Parallel Characterization Theorem [4]) *For every parallel algorithm, there is a behaviorally identical parallel ASM.*

The problem of characterizing distributed algorithms by suitable postulates is open.

4.2 ASMs and Hardware/Software Specifications

By the ASM thesis, ASMs are appropriate for modeling of arbitrary computer systems on given levels of abstraction. You can model existing or future systems; in other words, you can use ASMs to specify how hardware or software is supposed to function at a given level of abstraction. These specifications are executable. (Practical ASM languages typically use declarative means as well: preconditions, postconditions, invariants, and so on.) The executability of specification makes it much more useful. It allows you to address the following crucial questions.

oarieli@mta.ac.il

1. Does the specification satisfy the requirements?
2. Does the implementation satisfy the specification?

4.3 Experimental Confirmation of the ASM Thesis

A substantial amount of experimental confirmation of the thesis is found at the academic ASM website [1]; see also books [5,12]. In most cases people use ASMs not to check the thesis but to achieve their own goals; typically they use ASMs for modeling/specification purposes. But in the process they find out that ASMs suffice for their modeling purpose. In cases when Theorems 1 or 2 apply, a direct ASM simulation of a given piece of software or hardware may be more elegant than the generic simulation obtained from the proofs of Theorems 1 or 2.

One particularly impressive example of the ASM usage in academia is a large distributed ASM that gives the official dynamic semantics for SDL, the Specification and Description Language of the International Telecommunication Union [6]. Another impressive example is [12].

The use of ASMs at Microsoft is (very partially) reflected at [2].

5 AsmL, the ASM Language

ASMs are mathematical machines executable in principle. This is not good enough for applications. One needs practical engines to write down and execute ASMs. By the time I joined Microsoft, several ASM engines were in use. Siemens used ASM Workbench designed by Giuseppe Del Castillo at the University of Paderborn, Germany, as well as ASM Gopher designed by Joachim Schmid and Wolfram Schulte at the University of Ulm, Germany. Matthias Anlauff designed XASM at the Technical University of Berlin, Germany; since then Matthias moved to Kestrel, Palo Alto, CA, and XASM became an open-source ASM tool. More information about these and other ASM tools is found at [1].

Currently, the most powerful ASM engines are those developed by the Foundation of Software Engineering group at Microsoft Research. One of them is called AsmL, an allusion to ASM Language. AsmL can be downloaded from the AsmL website [2] and used for academic purposes. The site contains various auxiliary materials.

5.1 Features of AsmL

AsmL has a strong mathematical component. In particular, sets, sequences, maps and tuples are available as well as set comprehension $\{e(x) \mid x \in r \mid \phi(x)\}$, sequence comprehension and map comprehension.

AsmL is fully object oriented.

The crucial features of AsmL, intrinsic to ASMs, are massive synchronous parallelism and finite choice. ASMs steps are transactions, and in that sense AsmL programming is transaction programming.

oarieli@mta.ac.il

AsmL is fully integrated into the .NET framework which provides interoperability with great many languages and tools.

Literate programming via MS Word and automated programming via XML are provided. The demo, mentioned below, demonstrates literate programming among other things. The whole article [7] is in fact an AsmL document.

Here are some additional features of AsmL.

- Advanced type system: disjunctive types, semantic subtypes, generics,
- Pattern matching for structures and classes,
- Intra-step communication with outside world and among submachines,
- Reflection over execution,
- Data access, structural coverage,
- State as first class citizen,
- Processes (coming).

The AsmL compiler is written in AsmL.

5.2 Specifications vs. Prototypes

It is often argued that specifications are mere prototypes. Of course, specifications are prototypes but good specifications are more than that. They present a consistent high-level description of the system abstracting away irrelevant details. They describe what might happen and what must not happen. And they are not quickly destroyed and thrown away; instead they continue to serve as important documentation.

Here is an example that makes this point; the example comes with the AsmL distribution. The task is to specify in-place sorting that proceeds one swap at a time and always advances. Here is an AsmL program that does that.

```
var A as Seq of Integer = [3,1,2]

Swap()
   choose i,j in Indices(A)
      where i<j and A(i)>A(j)
         A(i) := A(j)
         A(j) := A(i)

Sort()
   step until fixpoint
      Swap()
```

The program is self-explanatory with one exception: the last two lines of the `Swap` procedure are executed in parallel so that there is no need to save the value of `A(i)`. In AsmL, parallelism is a default; you pay a syntactic price for sequentiality.

The sorting algorithm of the program is not efficient but it is the most general algorithm for the purpose. Any other in-place sorting algorithm that proceeds one swap at a time and always advances is a specialization of our algorithm.

oarieli@mta.ac.il

5.3 A Demo

A demonstration of AsmL is planned for the talk.

6 Requirements, Specifications, and Implementations

Consider the development of a new piece of software (or maybe a new version of an old piece). A product idea gives rise to a (typically informal) description of the product formulating various requirements that the product is supposed to satisfy. This description is the starting point for writing a design specification. Eventually the specification is implemented.

6.1 Does the Specification Satisfy the Requirements?

The question can be restated thus: how to debug the specification? Whether specification is declarative or executable, it is important that it is readable. But if the specification is executable, you can play out various scenarios. In the case of AsmL specification, given a few properties of the specification, the AsmL tool allows you to automatically derive a finite state machine that abstracts from other properties [7]. The finite state machine can be used to produce test suites and for model checking.

6.2 Does the Implementation Satisfy the Specification?

The question really is how to enforce the specification? To make the problem a bit more concrete, imagine that our product is just an API, that is an application programming interface, that reacts to particular actions.

If the specification is deterministic, run a sequence of actions on the API specification and record the reactions; the result can be used as an oracle against which to test the implementation or implementations.

However, specifications tend to be highly non-deterministic. The sorting specification above is a good example. You cannot use it to produce an oracle for conformance testing. To deal with this more general situation, a different and much more subtle approach is being used by the group of Foundations of Software Engineering [3]. We plan to illustrate the approach in the talk.

7 Postlude

We hope that the story of the ASM project will support the maxim that there is nothing more practical than good theory.

oarieli@mta.ac.il

References

1. The ASM Michigan Webpage, `http://www.eecs.umich.edu/gasm/`, maintained by James K. Huggins.
2. The AsmL webpage, `http://research.microsoft.com/foundations/AsmL/`.
3. Mike Barnett and Wolfram Schulte, "Runtime Verification of .NET Contracts", *Elsevier Journal of Systems and Software* 65:3 (2003), 199–208.
4. Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms," *ACM Transactions on Computational Logic* 4:4 (2003), 578–651.
5. Egon Börger and Robert Stärk, *Abstract State Machines*, Springer, 2003.
6. Uwe Glässer, Reinhard Gotzhein and Andreas Prinz, "Formal semantics of SDL-2000: Status and perspectives", *Computer Networks* 42:3 (2003), 343–358, Elsevier.
7. Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte and Margus Veanes, "Generating finite state machines from abstract state machines", ACM Software Engineering Notes 27:4 (2002), 112-122.
8. Yuri Gurevich, "Evolving algebras: An attempt to discover semantics", in G. Rozenberg and A. Salomaa, Editors, *Current Trends in Theoretical Computer Science*, World Scientific, 1993, 266–292.
9. Yuri Gurevich, "Evolving algebra 1993: Lipari guide", in E. Börger, Editor, *Specification and Validation Methods*, Oxford University Press, 1995, 9–36.
10. Yuri Gurevich, "For every sequential algorithm there is an equivalent sequential abstract state machine", ACM Transactions on Computational Logic, vol. 1, no. 1 2000), 77–111.
11. Yuri Gurevich and James K. Huggins, "The Semantics of the C programming language", Springer Lecture Notes in Computer Science 702 (1993), 274–308.
12. Robert F. Stärk, Joachim Schmid and Egon Börger, *Java and the Java Virtual Machine: Definition, Verification, Validation*, Springer, 2001.
13. Alan M. Turing, "On computable numbers, with an application to the Entscheidungsproblem", *Proceedings of London Mathematical Society*, series 2, vol. 42 (1936–1937), 230–265; correction, *ibidem*, vol. 43, 544–546. Available online at http://www.abelard.org/turpap2/tp2-ie.asp.

oarieli@mta.ac.il

Database Repair by Signed Formulae

Ofer Arieli[1], Marc Denecker[2], Bert Van Nuffelen[2], and Maurice Bruynooghe[2]

[1] Department of Computer Science, The Academic College of Tel-Aviv,
Antokolski 4, Tel-Aviv 61161, Israel
oarieli@mta.ac.il
[2] Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{marcd,bertv,maurice}@cs.kuleuven.ac.be

Abstract. We introduce a *simple* and *practically efficient* method for repairing inconsistent databases. The idea is to properly *represent* the underlying problem, and then use off-the-shelf applications for efficiently *computing* the corresponding solutions.
Given a possibly inconsistent database, we represent the possible ways to restore its consistency in terms of *signed formulae*. Then we show how the 'signed theory' that is obtained can be used by a variety of computational models for processing quantified Boolean formulae, or by constraint logic program solvers, in order to rapidly and efficiently compute desired solutions, i.e., consistent repairs of the database.

1 Introduction

In this paper we consider a uniform representation of repairs of inconsistent relational databases, that is, a general description of how to restore the consistency of databases instances that do not satisfy a given set of integrity constraints. We then show how this description can be used by a variety of computational methodologies for efficiently computing database *repairs*, i.e., new consistent database instances that differ from the original database instance by a minimal set of changes (with respect to set inclusion or set cardinality).

Reasoning with inconsistent databases has been extensively studied in the last few years, especially in the context of integrating (possibly contradicting) independent data-sources.[1] In this paper we introduce a novel representation of the repair problem as a theory that consists of what we call *signed formulae*. Then we illustrate how off-the-shelf computational systems can use the theory to solve the problem, i.e., to compute repairs of the database. Here we apply two types of tools for repairing a database:

– We show that the problem of finding repairs with minimal cardinality for a given database can be converted to the problem of finding minimal Herbrand models for the corresponding 'signed theory'. Thus, once the process

[1] See., e.g., [1,4,9,10,13,14,19,20,23] for more details on reasoning with inconsistent databases and further references to related works.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 14–30, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

for consistency restoration of the database has been represented by a signed theory (using a polynomial transformation), tools for minimal model computations (such as the Sicstus Prolog constraint solver [12], or the answer set programming solver dlv [15]) can be used to efficiently find the required repairs.

– For finding repairs that are minimal with respect to set inclusion, satisfiability solvers on appropriate quantified Boolean formulae (QBF) can be utilized. Again, we provide a polynomial-time transformation to (signed) QBF theories, and show how QBF solvers [5,11,16,17,18,21,26] can be used to restore the database consistency.

The rest of the paper is organized as follows: In the next section we formally define the underlying problem and in Section 3 we show how to represent it by signed formulae. In Sections 4 and 5 we show how constraint solvers for logic programs and quantified Boolean formulae can be utilized for computing database repairs based on the signed theories. In Section 6 we present some experimental results, and in Section 7 we conclude with some further remarks and observations.

2 Database Repairs

Let L be a first-order language, based on a fixed database schema S and a fixed domain D. Every element of D has a unique name. A *database instance* $\mathcal{D}$ consists of atoms in the language L that are instances of the schema S. As such, every database instance $\mathcal{D}$ has a finite active domain, $\mathcal{A}(\mathcal{D})$, which is a subset of D.

A *database* is a pair $(\mathcal{D}, \mathcal{IC})$, where $\mathcal{D}$ is a database instance, and $\mathcal{IC}$, the set of *integrity constraints*, is a finite and classically consistent set of formulae in L. Given a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$, we apply to it the closed word assumption, so only the facts that are explicitly mentioned in $\mathcal{D}$ are considered true. The underlying semantics of a database $(\mathcal{D}, \mathcal{IC})$ corresponds, therefore, to the *least Herbrand model* of $\mathcal{D}$ (notation: $\mathcal{H}^{\mathcal{D}}$), i.e., the model of $\mathcal{D}$ that assigns true to all the ground instances of atomic formulae in $\mathcal{D}$, and assigns false to all the other atoms.

Given a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$, let

$$\mathcal{DB}^{\mathcal{A}} = \mathcal{D} \cup \mathcal{IC}^{\mathcal{A}} = \mathcal{D} \cup \{\rho(\psi) \mid \psi \in \mathcal{IC},\ \rho : var(\psi) \to \mathcal{A}(\mathcal{D})\},$$

where ρ is a *ground substitution* of variables to the individuals of $\mathcal{A}(\mathcal{D})$, the active domain of $\mathcal{D}$.[2] $\mathcal{DB}^{\mathcal{A}}$ is called the *Herbrand expansion* of $\mathcal{DB}$. As $\mathcal{D}$, $\mathcal{IC}$, and $\mathcal{A}(\mathcal{D})$ are all finite sets, $\mathcal{DB}^{\mathcal{A}}$ is also finite, and so $\Sigma^{\mathcal{DB}} = \{p_1, p_2, \ldots, p_n\}$, the set of the (ground) atomic formulae that appear in $\mathcal{DB}^{\mathcal{A}}$, is finite as well. In

[2] Thus, e.g., $\rho(\forall x\, \psi(x)) = \psi(p_1) \wedge ... \wedge \psi(p_n)$ and $\rho(\exists x\, \psi(x)) = \psi(p_1) \vee ... \vee \psi(p_n)$, where $p_1, \ldots, p_n$ are the elements of $\mathcal{A}(\mathcal{D})$; the transformation for other formulae is standard.

oarieli@mta.ac.il

what follows we shall assume that the databases are grounded w.r.t. their active domains, therefore we shall omit the superscripts of $\mathcal{IC}^{\mathcal{A}}$ and $\mathcal{DB}^{\mathcal{A}}$.

We say that a formula ψ *follows* from a database instance $\mathcal{D}$ (notation: $\mathcal{D} \models \psi$) if the minimal Herbrand model of $\mathcal{D}$ is also a model of ψ. A database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ is *consistent* if every formula in $\mathcal{IC}$ follows from $\mathcal{D}$ (notation: $\mathcal{D} \models \mathcal{IC}$).[3]

Given a possibly inconsistent database, our goal is to restore its consistency, i.e., to 'repair' the database:

Definition 2.1. An *update* of a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ is a pair $(\mathsf{Insert}, \mathsf{Retract})$, s.t. $\mathsf{Insert} \cap \mathcal{D} = \emptyset$ and $\mathsf{Retract} \subseteq \mathcal{D}$.[4] A *repair* of $\mathcal{DB}$ is an update of $\mathcal{DB}$, for which $(\mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}, \mathcal{IC})$ is a consistent database.

Intuitively, a database is updated by inserting the elements of Insert and removing the elements of Retract. An update is a repair when the resulting database is consistent. Note that if $\mathcal{DB}$ is consistent, then $(\emptyset, \emptyset)$ is a repair of $\mathcal{DB}$.

Example 2.1. Let $\mathcal{DB} = \big(\{P(a)\}, \{\forall x(P(x) \rightarrow Q(x))\}\big)$. Clearly, this database is not consistent. The Herbrand expansion of $\mathcal{DB}$ is $(\{P(a)\}, \{P(a) \rightarrow Q(a)\})$, and it has three repairs, namely $\mathcal{R}_1 = (\{\}, \{P(a)\})$, $\mathcal{R}_2 = (\{Q(a)\}, \{\})$, and $\mathcal{R}_3 = (\{Q(a)\}, \{P(a)\})$ that correspond, respectively, to removing $P(a)$ from the database, inserting $Q(a)$ to the database, and performing both actions simultaneously.

Note that as the underlying semantics is determined by Herbrand interpretations, the Domain Closure Assumption[5] is implicit here, and should be regarded as another constraint that should be satisfied by every repair. Therefore, e.g., $(\{Q(b)\}, \{P(a)\})$ is *not* a repair of $\mathcal{DB}$ in this case, for any $b \neq a$. Another implicit assumption, induced by the use of Herbrand semantics, is that Clark's equality axioms are satisfied, and so the elements of $\Sigma^{\mathcal{DB}}$ are all different.

As the example above shows, there are many ways to repair a given database, some of them may not be very natural or sensible. It is usual, therefore, to specify some preference criterion on the possible repairs, and to apply only those that are *(most) preferred* with respect to the underlying criterion. The most common criteria for preferring a repair $(\mathsf{Insert}, \mathsf{Retract})$ over a repair $(\mathsf{Insert}', \mathsf{Retract}')$ are *set inclusion* [1,4,9,10,14,19,20], i.e.,

$(\mathsf{Insert}, \mathsf{Retract}) \leq_i (\mathsf{Insert}', \mathsf{Retract}')$, if $\mathsf{Insert} \cup \mathsf{Retract} \subseteq \mathsf{Insert}' \cup \mathsf{Retract}'$,

or *minimal cardinality* [4,13,23], i.e.,

$(\mathsf{Insert}, \mathsf{Retract}) \leq_c (\mathsf{Insert}', \mathsf{Retract}')$, if $|\mathsf{Insert}| + |\mathsf{Retract}| \leq |\mathsf{Insert}'| + |\mathsf{Retract}'|$.

[3] That is, there is no integrity constraint that is violated in $\mathcal{D}$.

[4] Note that by conditions (1) and (2) it follows that $\mathsf{Insert} \cap \mathsf{Retract} = \emptyset$.

[5] Namely, that the domain of every variable is in the set $\Sigma^{\mathcal{DB}}$ of the ground atoms that appear in $\mathcal{DB}$.

oarieli@mta.ac.il

Both criteria above reflect the intuitive feeling that a 'natural' way to repair an inconsistent database should require some minimal amount of changes, therefore the recovered data is kept 'as close as possible' to the original one. According to this view, for instance, each one of the repairs $\mathcal{R}_1$ and $\mathcal{R}_2$ of Example 2.1 is strictly better than $\mathcal{R}_3$. Note also, that $(\emptyset, \emptyset)$ is the *only* $\leq_i$-preferred and $\leq_c$-preferred repair of consistent databases, as expected.

3 Representation of Repairs by Signed Formulae

In what follows we represent (preferred) repairs in terms of what we call 'signed formulae'. Then we incorporate corresponding solvers in order to compute the repairs.

For every (ground) atom $p \in \Sigma^{\mathcal{DB}}$ we introduce a new atom, s_p, intuitively understood as 'switch p', or 'change the status of p', that is, s_p holds iff $p \in \mathsf{Insert} \cup \mathsf{Retract}$. For every integrity constraint $\psi \in \mathcal{IC}$ we define a new formulae, $\overline{\psi}$, obtained from ψ by simultaneously substituting every appearance of an atom p by a corresponding expression τ_p that is defined as follows:

$$\tau_p = \begin{cases} \neg s_p & \text{if } p \in \mathcal{D}, \\ s_p & \text{otherwise.} \end{cases}$$

The formula $\overline{\psi} = \psi[\tau_{p_1}/p_1, \ldots, \tau_{p_m}/p_m]$ (i.e., the simultaneous substitution in ψ of all the atomic formulae p_i, $1 \leq i \leq m$, by their 'signed expressions' τ_{p_i}) is called the *signed formula* that is obtained from ψ.

Given a repair $\mathcal{R} = (\mathsf{Insert}, \mathsf{Retract})$ of a database $\mathcal{DB}$, define a valuation $\nu^{\mathcal{R}}$ on $\{s_p \mid p \in \Sigma^{\mathcal{DB}}\}$ as follows:

$$\nu^{\mathcal{R}}(s_p) = t \text{ iff } p \in \mathsf{Insert} \cup \mathsf{Retract}.$$

$\nu^{\mathcal{R}}$ is called the valuation that is *associated with* $\mathcal{R}$. Conversely, a valuation ν on $\{s_p \mid p \in \Sigma^{\mathcal{DB}}\}$ *induces* a database update $\mathcal{R}^\nu = (\mathsf{Insert}, \mathsf{Retract})$, where $\mathsf{Insert} = \{p \notin \mathcal{D} \mid \nu(s_p) = t\}$ and $\mathsf{Retract} = \{p \in \mathcal{D} \mid \nu(s_p) = t\}$.[6] Obviously, these mappings are the inverse of each other.

Example 3.1. Let $\mathcal{DB} = (\{p\}, \{p \rightarrow q\})$ be a ground representation of the database considered in Example 2.1. In this case, the sign formula of $\psi = p \rightarrow q$ is $\overline{\psi} = \neg s_p \rightarrow s_q$, or, equivalently, $s_p \vee s_q$. Intuitively, this formula indicates that in order to restore the consistency of $\mathcal{DB}$, at least one of p or q should be 'switched', i.e., either p should be removed from the database or q should be inserted to it. Indeed, the three classical models of $\overline{\psi}$ are exactly the three valuations on $\{s_p, s_q\}$ that are associated with the three repairs of $\mathcal{DB}$ (see Example 2.1). The next theorem shows that this is not a coincidence.

[6] Clearly, $\mathcal{R}^\nu$ is an update of $\mathcal{DB}$, but it is not necessarily a repair of $\mathcal{DB}$ (see Definition 2.1).

oarieli@mta.ac.il

Theorem 3.1. *Let $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ be a database. Denote: $\overline{\mathcal{IC}} = \{\overline{\psi} \mid \psi \in \mathcal{IC}\}$.*

a) *if $\mathcal{R}$ is a repair of $\mathcal{DB}$ then $\nu^{\mathcal{R}}$ is a model of $\overline{\mathcal{IC}}$,*
b) *if ν is a model of $\overline{\mathcal{IC}}$ then $\mathcal{R}^{\nu}$ is a repair of $\mathcal{DB}$.*

Proof. For (a), suppose that $\mathcal{R}$ is a repair of $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$. Then, in particular, $\mathcal{D}^{\mathcal{R}} \models \mathcal{IC}$, where $\mathcal{D}^{\mathcal{R}} = \mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}$. Let $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}$ be the least Herbrand model of $\mathcal{D}^{\mathcal{R}}$, and let $\psi \in \mathcal{IC}$. Then $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(\psi) = t$, and so it remains to show that $\nu^{\mathcal{R}}(\overline{\psi}) = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(\psi)$. The proof of this is by induction on the structure of ψ, and we show only the base step (the rest is trivial), i.e., for every $p \in \Sigma^{\mathcal{DB}}$, $\nu^{\mathcal{R}}(\overline{p}) = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p)$. Indeed,

– $p \in \mathcal{D} \setminus \mathsf{Retract} \Rightarrow p \in \mathcal{D}^{\mathcal{R}} \Rightarrow \nu^{\mathcal{R}}(\overline{p}) = \nu^{\mathcal{R}}(\neg s_p) = \neg\nu^{\mathcal{R}}(s_p) = \neg f = t = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p)$.
– $p \in \mathsf{Retract} \Rightarrow p \in \mathcal{D} \setminus \mathcal{D}^{\mathcal{R}} \Rightarrow \nu^{\mathcal{R}}(\overline{p}) = \nu^{\mathcal{R}}(\neg s_p) = \neg\nu^{\mathcal{R}}(s_p) = \neg t = f = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p)$.
– $p \in \mathsf{Insert} \Rightarrow p \in \mathcal{D}^{\mathcal{R}} \setminus \mathcal{D} \Rightarrow \nu^{\mathcal{R}}(\overline{p}) = \nu^{\mathcal{R}}(s_p) = t = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p)$.
– $p \notin \mathcal{D} \cup \mathsf{Insert} \Rightarrow p \notin \mathcal{D}^{\mathcal{R}} \Rightarrow \nu^{\mathcal{R}}(\overline{p}) = \nu^{\mathcal{R}}(s_p) = f = \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p)$.

For part (b), suppose that ν is a model of $\overline{\mathcal{IC}}$. Let

$$\mathcal{R}^{\nu} = (\mathsf{Insert}, \mathsf{Retract}) = (\{p \notin \mathcal{D} \mid \nu(s_p) = t\}, \{p \in \mathcal{D} \mid \nu(s_p) = t\}).$$

We shall show that $\mathcal{R}^{\nu}$ is a repair of $\mathcal{DB}$. According to Definition 2.1, it is obviously an update. It remains to show that every $\psi \in \mathcal{IC}$ follows from $\mathcal{D}^{\mathcal{R}} = \mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}$, i.e., that $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(\psi) = t$, where $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}$ is the least Herbrand model of $\mathcal{D}^{\mathcal{R}}$. Since ν is a model of $\overline{\mathcal{IC}}$, $\nu(\overline{\psi}) = t$, and so it remains to show that $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(\psi) = \nu(\overline{\psi})$. Again, the proof is by induction on the structure of ψ, and we show only the base step, that is: for every $p \in \Sigma^{\mathcal{DB}}$, $\mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p) = \nu(\overline{p})$:

– $p \in \mathcal{D} \setminus \mathsf{Retract} \Rightarrow p \in \mathcal{D}^{\mathcal{R}}$, $\nu(s_p) = f \Rightarrow \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p) = t = \neg\nu(s_p) = \nu(\neg s_p) = \nu(\overline{p})$.
– $p \in \mathsf{Retract} \Rightarrow p \in \mathcal{D} \setminus \mathcal{D}^{\mathcal{R}}$, $\nu(s_p) = t$, $\Rightarrow \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p) = f = \neg\nu(s_p) = \nu(\neg s_p) = \nu(\overline{p})$.
– $p \in \mathsf{Insert} \Rightarrow p \in \mathcal{D}^{\mathcal{R}} \setminus \mathcal{D}$, $\nu(s_p) = t$, $\Rightarrow \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p) = t = \nu(s_p) = \nu(\overline{p})$.
– $p \notin \mathcal{D} \cup \mathsf{Insert} \Rightarrow p \notin \mathcal{D}^{\mathcal{R}}$, $\nu(s_p) = f$, $\Rightarrow \mathcal{H}^{\mathcal{D}^{\mathcal{R}}}(p) = f = \nu(s_p) = \nu(\overline{p})$. □

The last theorem implies, in particular, that in order to compute repairs for a given database $\mathcal{DB}$, it is sufficient to find the models of the signed formulae that are induced by the integrity constraints of $\mathcal{DB}$; the pairs that are induced by these models are the repairs of $\mathcal{DB}$.

Example 3.2. Consider again the (grounded) database of Examples 2.1 and 3.1. The corresponding signed formula $\overline{\psi} = s_p \vee s_q$ has three models $\{s_p : t, s_q : f\}$, $\{s_p : f, s_q : t\}$, and $\{s_p : t, s_q : t\}$.[7] These models induce, respectively, three pairs, $(\{\}, \{p\})$, $(\{q\}, \{\})$, and $(\{q\}, \{p\})$, which are the repairs of $\mathcal{DB}$ (cf. Example 2.1).

[7] We are denoting here by $p : x$ the fact that the atom p is assigned the value x by the corresponding valuation.

oarieli@mta.ac.il

4 Computing Preferred Repairs by Model Generation

In this section we show how solvers for constraint logic programs (CLPs), answer-set programming (ASP) and SAT solvers can be used for computing $\leq_c$-preferred repairs and $\leq_i$-preferred repairs. The experimental results are presented in Section 6.

4.1 Computing $\leq_c$-Preferred Repairs

By Theorem 3.1, the repairs of a database correspond exactly to the models of the signed theory. It is straightforward to see that $\leq_c$-preferred repairs of $\mathcal{DB}$ (i.e., those with minimal cardinality) correspond to models of $\overline{\mathcal{IC}}$ that minimize the number of t-assignments of the atoms s_p. Hence, the problem is to find Herbrand models for $\overline{\mathcal{IC}}$ with minimal cardinality (called $\leq_c$-minimal Herbrand models).

Theorem 4.1. *Let $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ be a database and $\overline{\mathcal{IC}} = \{\overline{\psi} \mid \psi \in \mathcal{IC}\}$. Then:*

a) *if $\mathcal{R}$ is a $\leq_c$-preferred repair of $\mathcal{DB}$, then $\nu^{\mathcal{R}}$ is a $\leq_c$-minimal Herbrand model of $\overline{\mathcal{IC}}$.*

b) *if ν is a $\leq_c$-minimal Herbrand model of $\overline{\mathcal{IC}}$, then $\mathcal{R}^{\nu}$ is a $\leq_c$-preferred repair of $\mathcal{DB}$.*

We discuss two techniques to compute $\leq_c$-minimal Herbrand models. The first approach is to use a finite domain CLP solver. Encoding the computation of $\leq_c$-preferred repair using a finite domain constraint solver is a straightforward process. The 'switch atoms' s_p are encoded as finite domain variables with domain $\{0, 1\}$. A typical encoding specifies the relevant constraints (i.e., the encoding of $\overline{\mathcal{IC}}$), assigns a special variable, `Sum`, for summing-up all the signed variables that are assigned the value '1', and asks for a solution with a minimal value for `Sum`.

Example 4.1. Below is a code for repairing the database of Example 3.2 with Sicstus Prolog finite domain constraint solver CLP(FD) [12][8].

```
domain([Sp,Sq],0,1),                  % domain of the signed atoms
Sp #\/ Sq,                            % the signed theory
sum([Sp,Sq],#=,Sum),                  % Sum = num of vars with val 1
minimize(labeling([],[Sp,Sq]),Sum). % find a solution with min sum
```

The solutions computed here are $[1, 0]$ and $[0, 1]$, and the value of `Sum` is 1. This means that the cardinality of the $\leq_c$-preferred repairs of $\mathcal{DB}$ should be 1, and that these repairs are induced by the valuations $\nu_1 = \{s_p : t, s_q : f\}$ and $\nu_2 = \{s_p : f, s_q : t\}$. Thus, the two $\leq_c$-minimal repairs here are $(\{\}, \{p\})$ and $(\{q\}, \{\})$, which indeed insert or retract exactly one atomic formula.

[8] A Boolean constraint solver would also be appropriate here. As Sicstus Prolog Boolean constraint solver has no minimization capabilities, we prefer to use here the finite domain constraint solver.

oarieli@mta.ac.il

A second approach is to use the disjunctive logic programming system DLV [15]. To compute $\leq_c$-minimal repairs using DLV, the signed theory $\overline{\mathcal{IC}}$ is transformed into a propositional clausal form. A clausal theory is a special case of a disjunctive logic program without negation in the body of the clauses. The stable models of a disjunctive logic program without negation as failure in the body of rules coincide exactly with the $\leq_i$-minimal models of such a program. Hence, by transforming the signed theory $\overline{\mathcal{IC}}$ to clausal form, DLV can be used to compute $\leq_i$-minimal Herbrand models. To eliminate models with non-minimal cardinality, *weak constraints* are used. A weak constraint is a formula for which a cost value is defined. With each model computed by DLV, a cost is defined as the sum of the cost values of all weak constraints satisfied in the model. The DLV system can be asked to generate models with minimal total cost. The set of weak constraints used to compute $\leq_c$-minimal repairs is exactly the set of all atoms s_p; each atom has cost 1. Clearly, $\leq_i$-minimal models of a theory with minimal total cost are exactly the models with least cardinality.

Example 4.2. Below is a code for repairing the database of Example 3.2 with DLV.

```
Sp v Sq.       % the clause
:~ Sp.         % the weak constraints (their cost is 1 by default)
:~ Sq.
```

Clearly, the solutions here are $\{s_p : t, s_q : f\}$ and $\{s_p : f, s_q : t\}$. These valuations induce the two $\leq_c$-minimal repairs of $\mathcal{DB}$, $\mathcal{R}_1 = (\{\}, \{p\})$ and $\mathcal{R}_2 = (\{q\}, \{\})$.

4.2 Computing $\leq_i$-Preferred Repairs

The $\leq_i$-preferred repairs of a database correspond to minimal Herbrand models with respect to set inclusion of the signed theory $\overline{\mathcal{IC}}$. We focus on the computation of one minimal model. The reason is simply that in most sizable applications, the computation of all minimal models is not feasible (there are too many of them). We consider here three simple techniques to compute a $\leq_i$-preferred repair. In the next section we consider another more complex method.

I. One technique, mentioned already in the previous section, is to transform $\overline{\mathcal{IC}}$ to clausal form and use the DLV system. In this case the weak constraints are not needed.

II. Another possibility is to adapt CLP-techniques to compute $\leq_i$-minimal models of Boolean constraints. The idea is simply to make sure that whenever a Boolean variable (or a finite domain variable with domain $\{0, 1\}$) is selected for being assigned a value, one first assigns the value 0 before trying to assign the value 1.

Proposition 4.1. *If the above strategy for value selection is used, then the first computed model is provably a $\leq_i$-minimal model.*

oarieli@mta.ac.il

Proof. Consider the search tree of the CLP-problem. Each path in this tree represents a value assignment to a subset of the constraint variables. Internal nodes, correspond to partial solutions, are labeled with the variable selected by the labeling function of the solver and have two children: the left child assigns value 0 to the selected variable and the right child assigns value 1. We say that node n_2 is on the right of a node n_1 in this tree if n_2 appears in the right subtree, and n_1 appears in the left subtree of the deepest common ancestor node of n_1 and n_2. It is then easy to see that in such a tree, each node n_2 to the right of a node n_1 assigns the value 1 to the variable selected in this ancestor node, whereas n_1 assigns value 0 to this variable. Consequently, the left-most node in the search tree which is a model of the Boolean constraints, is $\leq_i$-minimal. □

In CLP-systems such as Sicstus Prolog, one can control the order in which values are assigned to variables. We have implemented the above strategy and discuss the results in Section 6.

III. A third technique considered here uses SAT-solvers. SAT-solvers, such as zChaff [25], do not compute directly minimal models, but can be easily extended to do so. The algorithm uses the SAT-solver to generate models of the theory $\mathcal{T}$, until it finds a minimal model. Minimality of a model M of $\mathcal{T}$ can be verified by checking the unsatisfiability of $\mathcal{T}$, augmented with the axioms $\bigvee_{p \in M} \neg p$ and $\bigwedge_{p \notin M} \neg p$. The model M is minimal exactly when these axioms are inconsistent with $\mathcal{T}$. This approach has been tested using the SAT solver zChaff [25]; the results are discussed in Section 6.

5 Computing $\leq_i$-Preferred Repairs by QBF Solvers

In this section we show how solvers for quantified Boolean formulae (QBFs) can be used for computing the $\leq_i$-preferred repairs of a given database. In this case it is necessary to add to the signed formulae of $\overline{\mathcal{IC}}$ an axiom (represented by a quantified Boolean formula) that expresses $\leq_i$-minimality, i.e., that an $\leq_i$-preferred repair is not included in any other database repair. Then, QBF solvers such as QUBOS [5], EVALUATE [11], QUIP [16], QSOLVE [17], QuBE [18], QKN [21], SEMPROP [22], and DECIDE [26], can be applied to the signed quantified Boolean theory that is obtained, in order to compute the $\leq_i$-preferred repairs of the database. Below we give a formal description of this process.

5.1 Quantified Boolean Formulae

Quantified Boolean formulae (QBFs) are propositional formulae extended with quantifiers $\forall, \exists$ over propositional variables. In what follows we shall denote propositional formulae by Greek lower-case letters (usually ψ, ϕ) and QBFs by Greek upper-case letters (e.g., Ψ, Φ). Intuitively, the meaning of a QBF of the form $\exists p \, \forall q \, \psi$ is that there exists a truth assignment of p such that ψ is true for every truth assignment of q. Next we formalize this intuition.

oarieli@mta.ac.il

As usual, we say that an occurrence of an atomic formula p is *free* if it is not in the scope of a quantifier $\mathsf{Q}p$, for $\mathsf{Q} \in \{\forall, \exists\}$, and we denote by $\Psi[\phi_1/p_1, \ldots, \phi_m/p_m]$ the uniform substitution of each free occurrence of a variable p_i in Ψ by a formula ϕ_i, for $i = 1, \ldots, m$. The notion of a *valuation* is extended to QBFs as follows: Given a function $\nu_{\rm at} : \Sigma^{\mathcal{DB}} \cup \{\mathsf{t}, \mathsf{f}\} \to \{t, f\}$ s.t. $\nu(\mathsf{t}) = t$ and $\nu(\mathsf{f}) = f$, a valuation ν on QBFs is recursively defined as follows:

$\nu(p) = \nu_{\rm at}(p)$ for every $p \in \Sigma^{\mathcal{DB}} \cup \{\mathsf{t}, \mathsf{f}\}$,
$\nu(\neg\psi) = \neg\nu(\psi)$,
$\nu(\psi \circ \phi) = \nu(\psi) \circ \nu(\phi)$, where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$,
$\nu(\forall p\, \psi) = \nu(\psi[\mathsf{t}/p]) \wedge \nu(\psi[\mathsf{f}/p])$,
$\nu(\exists p\, \psi) = \nu(\psi[\mathsf{t}/p]) \vee \nu(\psi[\mathsf{f}/p])$.

A valuation ν *satisfies* a QBF Ψ if $\nu(\Psi) = t$; ν is a *model* of a set Γ of QBFs if it satisfies every element of Γ. A QBF Ψ is *entailed by* a set Γ of QBFs (notation: $\Gamma \vdash \Psi$) if every model of Γ is also a model of Ψ. In what follows we shall use the following notations: for two valuations ν_1 and ν_2 we denote by $\nu_1 \leq \nu_2$ that for every atomic formula p, $\nu_1(p) \rightarrow \nu_2(p)$ is true. We shall also write $\nu_1 < \nu_2$ to denote that $\nu_1 \leq \nu_2$ and $\nu_2 \not\leq \nu_1$.

5.2 Representing $\leq_i$-Preferred Repairs by Signed QBFs

It is well-known that quantified Boolean formulae can be used for representing circumscription [24], thus they properly express logical minimization [7,8]. In our case we use this property for expressing minimization of repairs w.r.t. set inclusion.

Given a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$, denote by $\overline{\mathcal{IC}}_\wedge$ the conjunction of all the elements in $\overline{\mathcal{IC}}$ (i.e., the conjunction of all the signed formulae that are obtained from the integrity constraints of $\mathcal{DB}$). Consider the following QBF, denoted $\Psi_{\mathcal{DB}}$:

$$\forall s'_{p_1}, \ldots, s'_{p_n} \Big(\overline{\mathcal{IC}}_\wedge \left[s'_{p_1}/s_{p_1}, \ldots, s'_{p_n}/s_{p_n} \right] \rightarrow \big(\bigwedge_{i=1}^{n} (s'_{p_i} \rightarrow s_{p_i}) \rightarrow \bigwedge_{i=1}^{n} (s_{p_i} \rightarrow s'_{p_i}) \big) \Big).$$

Consider a model ν of $\overline{\mathcal{IC}}_\wedge$, i.e., a valuation for $s_{p_1}, \ldots, s_{p_n}$ that makes $\overline{\mathcal{IC}}_\wedge$ true. The QBF $\Psi_{\mathcal{DB}}$ expresses that every interpretation μ (valuation for $s'_{p_1}, \ldots, s'_{p_n}$) that is a model of $\overline{\mathcal{IC}}_\wedge$, has the property that $\mu \leq \nu$ implies $\nu \leq \mu$, i.e., there is no model μ of $\overline{\mathcal{IC}}_\wedge$, s.t. the set $\{s_p \mid \nu(s_p) = t\}$ properly contains the set $\{s_p \mid \mu(s_p) = t\}$. In terms of database repairs, this means that if $\mathcal{R}^\nu = (\mathsf{Insert}, \mathsf{Retract})$ and $\mathcal{R}^\mu = (\mathsf{Insert}', \mathsf{Retract}')$ are the database repairs that are associated, respectively, with ν and μ, then $\mathsf{Insert}' \cup \mathsf{Retract}' \not\subset \mathsf{Insert} \cup \mathsf{Retract}$. It follows, therefore, that in this case $\mathcal{R}^\nu$ is a $\leq_i$-preferred repair of $\mathcal{DB}$, and in general $\Psi_{\mathcal{DB}}$ represents $\leq_i$-minimality.

Example 5.1. With the database $\mathcal{DB}$ of Examples 2.1, 3.1, and 3.2, $\overline{\mathcal{IC}} \cup \Psi_{\mathcal{DB}}$ is the following theory, Γ:

$$\Big\{ s_p \vee s_q \ ,\ \forall s'_p \forall s'_q \Big((s'_p \vee s'_q) \rightarrow ((s'_p \rightarrow s_p) \wedge (s'_q \rightarrow s_q) \ \rightarrow \ (s_p \rightarrow s'_p) \wedge (s_q \rightarrow s'_q)) \Big) \Big\}.$$

oarieli@mta.ac.il

The models of Γ are those that assign t either to s_p or to s_q, but not to both of them, i.e., $\nu_1 = (s_p : t, s_q : f)$ and $\nu_2 = (s_p : f, s_q : t)$. The database updates that are induced by these valuations are, respectively, $\mathcal{R}^{\nu_1} = (\{\}, \{p\})$ and $\mathcal{R}^{\nu_2} = (\{q\}, \{\})$. By Theorem 5.1 below, these are the only $\leq_i$-preferred repairs of $\mathcal{DB}$.

Theorem 5.1. *Let $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ be a database and $\overline{\mathcal{IC}} = \{\overline{\psi} \mid \psi \in \mathcal{IC}\}$. Then:*

a) *if $\mathcal{R}$ is an $\leq_i$-preferred repair of $\mathcal{DB}$ then $\nu^{\mathcal{R}}$ is a model of $\overline{\mathcal{IC}} \cup \Psi_{\mathcal{DB}}$,*
b) *if ν is a model of $\overline{\mathcal{IC}} \cup \Psi_{\mathcal{DB}}$ then $\mathcal{R}^{\nu}$ is an $\leq_i$-preferred repair of $\mathcal{DB}$.*

Proof. Suppose that $\mathcal{R} = (\mathsf{Insert}, \mathsf{Retract})$ is an $\leq_i$-preferred repair of $\mathcal{DB}$. In particular, it is a repair of $\mathcal{DB}$ and so, by Theorem 3.1, $\nu^{\mathcal{R}}$ is a model of $\overline{\mathcal{IC}}$. Since Theorem 3.1 also assures that a database update that is induced by a model of $\overline{\mathcal{IC}}$ is a repair of $\mathcal{DB}$, in order to prove both parts of the theorem, it remains to show that the fact that $\nu^{\mathcal{R}}$ satisfies $\Psi_{\mathcal{DB}}$ is a necessary and sufficient condition for assuring that $\mathcal{R}$ is $\leq_i$-minimal among the repairs of $\mathcal{DB}$. Indeed, $\nu^{\mathcal{R}}$ satisfies $\Psi_{\mathcal{DB}}$ iff for every valuation μ that satisfies $\overline{\mathcal{IC}}_{\wedge}$ and for which $\mu \leq \nu^{\mathcal{R}}$, it is also true that $\nu^{\mathcal{R}} \leq \mu$. Thus, $\nu^{\mathcal{R}}$ satisfies $\Psi_{\mathcal{DB}}$ iff there is no model μ of $\overline{\mathcal{IC}}$ s.t. $\mu < \nu^{\mathcal{R}}$, iff (by Theorem 3.1 again) there is no repair $\mathcal{R}'$ of $\mathcal{DB}$ s.t. $\nu^{\mathcal{R}'} < \nu^{\mathcal{R}}$, iff there is no repair $\mathcal{R}' = (\mathsf{Insert}', \mathsf{Retract}')$ s.t. $\mathsf{Insert}' \cup \mathsf{Retract}' \subset \mathsf{Insert} \cup \mathsf{Retract}$, iff $\mathcal{R}$ is an $\leq_i$-minimal repairs of $\mathcal{DB}$. □

Note 5.1. (*Complexity results*) A skeptical (conservative) approach to query answering is considered, e.g., in [1,19], where an answer to a query $\mathcal{Q}$ and a database $\mathcal{DB}$ is evaluated with respect to (the databases that are obtained from) *all* the $\leq_i$-preferred repairs of $\mathcal{DB}$. A credulous approach to the same problem evaluates queries with respect to *some* $\leq_i$-preferred repair of $\mathcal{DB}$. Theorem 5.1 implies the following upper complexity bounds for these approaches:

Corollary 5.1. *Credulous query answering lies in Σ_2^P, and skeptical query answering is in Π_2^P.*

Proof. By Theorem 5.1, credulous query answering is equivalent to satisfiability checking for $\overline{\mathcal{IC}} \cup \Psi_{\mathcal{DB}}$, and conservative query answering is equivalent to entailment checking for the same theory (see also Corollary 5.2 below). Thus, these decision problems can be encoded by QBFs in prenex normal form with exactly one quantifier alternation. The corollary is obtained, now, by the following well-known result:

Proposition 5.1. [27] *Given a propositional formula ψ, whose atoms are partitioned into $i \geq 1$ sets $\{p_1^1, \ldots, p_{m_1}^1\}, \ldots, \{p_1^i, \ldots, p_{m_i}^i\}$, deciding whether*

$$\exists p_1^1, \ldots, \exists p_{m_1}^1, \forall p_1^2, \ldots, \forall p_{m_2}^2, \ldots, \mathsf{Q} p_1^i, \ldots, \mathsf{Q} p_{m_i}^i \psi$$

is true, is Σ_i^P-complete (where $\mathsf{Q} = \exists$ if i is odd and $\mathsf{Q} = \forall$ if i is even). Also, deciding if

$$\forall p_1^1, \ldots, \forall p_{m_1}^1, \exists p_1^2, \ldots, \exists p_{m_2}^2, \ldots, \mathsf{Q} p_1^i, \ldots, \mathsf{Q} p_{m_i}^i \psi$$

is true, is Π_i^P-complete (where $\mathsf{Q} = \forall$ if i is odd and $\mathsf{Q} = \exists$ if i is even). □

oarieli@mta.ac.il

As shown, e.g., in [19], the complexity bounds specified in the last corollary are strict, i.e., these decision problems are hard for the respective complexity classes.

Note 5.2. (*Consistent query answering*) Another consequence of Theorem 5.1 is that the conservative approach to query answering [1,19] may be represented in our context in terms of a consequence relation as follows:

Corollary 5.2. *$\mathcal{Q}$ is a consistent query answer of a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ in the sense of* [1,19] *iff $\overline{\mathcal{IC}} \cup \Psi_{\mathcal{DB}} \vdash \mathcal{Q}$.*

The last corollary and Section 4.2 provide, therefore, some additional methods for consistent query answering, all of them are based on signed theories.

6 Experiments and Comparative Study

The idea of using formulae that introduce new ('signed') variables aimed at designating the truth assignments of other related variables is used, for different purposes, e.g. in [2,3,6,7]. In the area of database integration, signed variables are used in [19], and have a similar intended meaning as in our case. In [19], however, only $\leq_i$-preferred repairs are considered, and a rewriting process for converting relational queries over a database with constraints to extended disjunctive queries (with two kinds of negations) over database without constraints, must be employed. As a result, only solvers that are able to process disjunctive Datalog programs and compute their stable models (e.g., DLV), can be applied. In contrast, as we have already noted above, motivated by the need to find *practical* and *effective* methods for repairing inconsistent databases, signed formulae serve here as a representative platform that can be directly used by a variety of off-the-shelf applications for computing (either $\leq_i$-preferred or $\leq_c$-preferred) repairs. In what follows we examine some of these applications and compare their appropriateness to the kind of problems that we are dealing with.

We have randomly generated instances of a database, consisting of three relations: *teacher* of the schema (`teacher_name`), *course* of the schema (`course_name`), and *teaches* of the schema (`teacher_name`, `course_name`). Also, the following two integrity constraints were specified:

ic1 A course is given by one teacher:

$$\forall X\, \forall Y\, \forall Z \Big(\big(\mathit{teacher}(X) \ \wedge \ \mathit{teacher}(Y) \ \wedge \ \mathit{course}(Z) \wedge \ \mathit{teaches}(X,Z) \wedge \mathit{teaches}(Y,Z) \big) \ \rightarrow \ X = Y \Big)$$

ic2 Each teacher gives at least one course:

$$\forall X \Big(\mathit{teacher}(X) \ \rightarrow \ \exists Y \big(\mathit{course}(Y) \wedge \mathit{teaches}(X,Y) \big) \Big)$$

The next four test cases (identified by the enumeration below) were considered:

oarieli@mta.ac.il

1. Small database instances with **ic1** as the only constraint.
2. Larger database instances with **ic1** as the only constraint.
3. Databases with $\mathcal{IC} = \{\mathbf{ic1}, \mathbf{ic2}\}$, where the number of courses equals the number of teachers.
4. Databases with $\mathcal{IC} = \{\mathbf{ic1}, \mathbf{ic2}\}$ and with fewer courses than teachers.

Note that in the first two test cases, only retractions of database facts are needed in order to restore consistency, in the third test case both insertion and retractions may be needed, and the last test case is unsolvable, as the theory is not satisfiable.

For each benchmark we generated a sequence of instances with an increasing number of database facts, and tested them w.r.t. the following applications:

– **ASP/CLP-solvers:**
DLV [15] (release 2003-05-16), CLP(FD) [12] (version 3.10.1).
– **QBF-solvers:**
SEMPROP [22] (release 24.02.02), QuBE-BJ [18] (release 1.3), DECIDE [26].
– **SAT-solvers:**
A minimal-model generator based on zChaff [25].

The goal was to construct $\leq_i$-preferred repairs within a time limit of five minutes. The systems DLV and CLP(FD) were tested also for constructing $\leq_c$-preferred repairs. All the experiments were done on a Linux machine, 800MHz, with 512MB memory. Tables 1–4 show the results for providing the first answer.[9]

The results of the first benchmark (Table 1) already indicate that DLV, CLP, and zChaff perform much better than the QBF-solvers. In fact, among the QBF-solvers that were tested, only SEMPROP could repair within the time limit most of the database instances of benchmark 1, and none of them could successfully repair (within the time restriction) the larger database instances, tested in benchmark 2. Also, we encountered some space limitation problems and a bug[10] in DECIDE, and this discouraged us from using it in our experiments.

Another observation from Tables 1–4 is that DLV, CLP, and the zChaff-based system, perform very good for minimal inclusion greedy algorithms. However, when using DLV and CLP for cardinality minimization, their performance is much worse. This is due to an exhaustive search for a $\leq_c$-minimal solution.

While in benchmark 1 the time differences among DLV, CLP, and zChaff, for computing $\leq_i$-repairs are marginal, in the other benchmarks the differences

[9] Times are in given in seconds, empty cells mean that timeout is reached without an answer, vars is the number of variables, IC is the number of grounded integrity constraints, and size is the size of the repairs.

[10] For the unsatisfiable QBF $\exists xy \forall uv((x \vee y) \wedge (u \vee v))$, the answer $x = 1$ and $y = 0$ is returned. The system developers were notified about this and the bug is being fixed.

oarieli@mta.ac.il

Table 1. Results for test case 1

Test info.			$\leq_i$-repairs						$\leq_c$-repairs	
No.	vars	IC	size	DLV	CLP	zChaff	SEMPROP	QuBE	DLV	CLP
1	20	12	8	0.005	0.010	0.024	0.088	14.857	0.011	0.020
2	25	16	7	0.013	0.010	0.018	0.015		0.038	0.020
3	30	28	12	0.009	0.020	0.039	0.100		0.611	0.300
4	35	40	15	0.023	0.020	0.008	0.510		2.490	1.270
5	40	48	16	0.016	0.020	0.012	0.208		3.588	3.220
6	45	42	17	0.021	0.030	0.008	0.673		12.460	10.350
7	50	38	15	0.013	0.020	0.009	0.216		23.146	20.760
8	55	50	20	0.008	0.030	0.018	1.521		29.573	65.530
9	60	58	21	0.014	0.030	0.036	3.412		92.187	136.590
10	65	64	22	0.023	0.030	0.009	10.460		122.399	171.390
11	70	50	22	0.014	0.030	0.019	69.925			
12	75	76	27	0.021	0.030	0.010	75.671			
13	80	86	29	0.021	0.030	0.009	270.180			
14	85	76	30	0.022	0.030	0.010				
15	90	78	32	0.024	0.040	0.020				
16	95	98	35	0.027	0.040	0.047				
17	100	102	40	0.017	0.040	0.016				
18	105	102	37	0.018	0.040	0.033				
19	110	124	43	0.030	0.040	0.022				
20	115	116	44	0.027	0.040	0.041				

Table 2. Results for test case 2

Test info.			$\leq_i$-repairs			
No.	vars	IC	size	DLV	CLP	zChaff
1	480	171	470	0.232	0.330	0.155
2	580	214	544	0.366	0.440	0.051
3	690	265	750	0.422	0.610	0.062
4	810	300	796	0.639	0.860	0.079
5	940	349	946	0.815	1.190	0.094
6	1080	410	1108	1.107	1.560	0.123
7	1230	428	1112	1.334	2.220	0.107
8	1390	509	1362	1.742	2.580	0.135
9	1560	575	1562	2.254	3.400	0.194
10	1740	675	1782	2.901	4.140	0.182
11	1930	719	2042	3.592	5.260	0.253

become more evident. Thus, for instance, zChaff performs better than the other solvers w.r.t. bigger database instances with many simple constraints (see benchmark 2), while DLV performs better when the problem has bigger and more complicated sets of constraints (see benchmark 3). The SAT approach with zChaff was the fastest in detecting unsatisfiable situations (see benchmark 4). As shown in Table 4, detecting unsatisfiability requires a considerable amount of time, even for small instances.

oarieli@mta.ac.il

Table 3. Results for test case 3

Test info.		$\leq_i$-repairs				$\leq_c$-repairs	
No.	vars	size	DLV	CLP	zChaff	DLV	CLP
1	25	4	0.008	0.030	0.066	0.010	0.05
2	36	9	0.008	0.030	0.087	0.070	0.42
3	49	15	0.027	0.250	0.050	0.347	9.48
4	64	23	0.019	0.770	0.013	2.942	58.09
5	81	30	0.012	4.660	0.102	26.884	
6	100	34	0.021		0.058	244.910	
7	121	38	0.626		1.561		
8	144	47	0.907		2.192		
9	169	51	0.161		0.349		
10	196	68	1.877		4.204		
11	225	70	8.496		16.941		

Table 4. Results for test case 4

Test info.			$\leq_i$-repairs			$\leq_c$-repairs	
No.	teachers	courses	DLV	CLP	zChaff	DLV	CLP
1	5	4	0.001	0.01	0.001	0.001	0.001
2	7	5	0.005	0.13	0.010	0.005	0.120
3	9	6	0.040	1.41	0.020	0.042	1.400
4	11	7	0.396	17.18	0.120	3.785	17.170
5	13	8	3.789		1.050	44.605	
6	15	9	44.573		13.370		
7	17	10					

Some of the conclusions from the experiments may be summarized as follows:

1. In principle, QBF-solvers, CLP-solvers, ASP-solvers, and SAT-solvers are all adequate tools for computing database repairs.
2. All the QBF-solvers, as well as DLV and zChaff, are 'black-boxes' that accept the problem specification in a certain format. In contrast, CLP(FD) provides a more 'open' environment, in which it is possible to incorporate problem-specific search algorithms, such as the greedy algorithm for finding $\leq_i$-minimal repairs (see Section 4.2).
3. Currently, the performance of the QBF-solvers is considerably below that of the other solvers. Moreover, most of the QBF-solvers require that the formulae are represented in prenex CNF, and specified in Dimacs or Rintanen format. These requirements are usually space-demanding. In our context, the fact that many QBF-solvers (e.g., SEMPROP and QuBE-BJ) return only yes/no answers (according to the satisfiability of the input theory), is another problem, since it is impossible to construct repairs only by these answers.

oarieli@mta.ac.il

One needs to be able to extract the assignments to the outmost existentially quantified variables (as done, e.g., by DECIDE).
Despite these drawbacks of QBF-solvers, reasoning with QBFs seems to be particularly suitable for our needs, since this framework provides a natural way to express minimization (in our case, representations of optimal repairs). It is most likely, therefore, that future versions of QBF-solvers will be the basis of powerful mechanisms for handling consistency in databases.

7 Concluding Remarks

This work provides further evidence for the well-known fact that in many cases a proper representation of a given problem is a major step in finding robust solutions to it. In our case, a uniform method for encoding the restoration of database consistency by signed formulae allows us to use off-the-shelf solvers for efficiently computing the desired repairs.

As shown in Corollary 5.1, the task of repairing a database is on the second level of the polynomial hierarchy, hence it is not tractable. However, despite the high computational complexity of the problem, the experimental results of Section 6 show that our method of repairing databases by signed theories is *practically appealing*, as it allows a rapid construction of repairs for large problem instances.

References

1. M.Arenas, L.Bertossi, and J.Chomicki. Consistent query answers in inconsistent databases. *Proc. 18th ACM Symp. on Principles of Database Systems* (PODS'99), pp.68–79, 1999.
2. O.Arieli and M.Denecker. Modeling paraconsistent reasoning by classical logic. *Proc. 2nd Symp. on Foundations of Information and Knowledge Systems* (FoIKS'02), T.Eiter and K.D.Schewe, editors, LNCS 2284, Springer, pp.1–14, 2002.
3. O.Arieli and M.Denecker. Reducing preferential paraconsistent reasoning to classical entailment. *Journal of Logic and Computation* 13(4), pp.557–580, 2003.
4. O.Arieli, B.Van Nuffelen, M.Denecker, and M.Bruynooghe. Coherent composition of distributed knowledge-bases through abduction. *Proc. 8th Int. Conf. on Logic Programming, Artificial Intelligence and Reasoning* (LPAR'01), A.Nieuwenhuis and A.Voronkov, editors, LNCS 2250, Springer, pp.620–635, 2001.
5. A.Ayari and D.Basin. QUBOS: Deciding quantified Boolean logic using propositional satisfiability solvers. *Proc. 4th Int. Conf. on Formal Methods in Computer-Aided Design* (FMCAD'02), M.D.Aagaard and J.W.O'Leary, editors, LNCS 2517, Springer, pp.187–201, 2002.
6. P.Besnard, T.Schaub. Signed systems for paraconsistent reasoning. *Journal of Automated Reasoning* 20(1), pp.191–213, 1998.
7. P.Besnard, T.Schaub, H.Tompits, and S.Woltran. Paraconsistent reasoning via quantified Boolean formulas, part I: Axiomatizing signed systems. *Proc. 8th European Conf. on Logics in Artificial Intelligence* (JELIA'02), S.Flesca et al., editors, LNAI 2424, Springer, pp.320–331, 2002.

oarieli@mta.ac.il

8. P.Besnard, T.Schaub, H.Tompits, and S.Woltran. Paraconsistent reasoning via quantified Boolean formulas, part II: Circumscribing inconsistent theories. *Proc. 7th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (ECSQARU'03), T.D.Nielsen and N.L.Zhang, editors, LNAI 2711, Springer, pp.528–539, 2003.
9. L.Bertossi, J.Chomicki, A.Cortes, and C.Gutierrez. Consistent answers from integrated data sources. *Proc. Flexible Query Answering Systems* (FQAS'2002), A.Andreasen et al., editors, LNCS 2522, Springer, pp.71–85, 2002.
10. L.Bertossi and C.Schwind. Analytic tableau and database repairs: Foundations. *Proc. 2nd Int. Symp. on Foundations of Information and Knowledge Systems* (FoIKS'02), T.Eiter and K.D.Schewe, editors, LNCS 2284, Springer, pp.32–48, 2002.
11. M.Cadoli, M.Schaerf, A.Giovanardi, and M.Giovanardi. An Algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *Automated Reasoning* 28(2), pp.101–142, 2002.
12. M.Carlsson, G.Ottosson and B.Carlson. An open-ended finite domain constraint solver, *Proc. 9th Int. Symp. on Programming Languages, Implementations, Logics, and Programs* (PLILP'97), LNCS 1292, Springer, 1997.
13. M.Dalal. Investigations into a theory of knowledge base revision. *Proc. National Conference on Artificial Intelligence* (AAAI'98), AAAI Press, pp.475–479, 1988.
14. S.de Amo, W.Carnielli, and J.Marcos. A logical framework for integrating inconsistent information in multiple databases. *Proc. 2nd Int. Symp. on Foundations of Information and Knowledge Systems* (FoIKS'02), T.Eiter and K.D.Schewe, editors, LNCS 2284, Springer, pp.67–84, 2002.
15. T.Eiter, N.Leone, C.Mateis, G.Pfeifer, and F.Scarcello. The KR system dlv: Progress report, comparisons and benchmarks. *Proc. 6th Int. Conf. on Principles of Knowledge Representation and Reasoning* (KR'98), Morgan Kaufmann Publishers, pp.406–417, 1998.
16. U.Egly, T.Eiter, H.Tompits, and S.Woltran. Solving advanced reasoning tasks using quantified Boolean formulas. *Proc. National Conf. on Artificial Intteligence* (AAAI'00), AAAI Press, pp.417–422, 2000.
17. R.Feldmann, B.Monien, and S.Schamberger. A distributed algorithm to evaluate quantified Boolean formulae. *Proc. National Conf. on Artificial Intteligence* (AAAI'00), AAAI Press, pp. 285–290, 2000.
18. E.Giunchiglia, M.Narizzano, and A.Tacchella. QuBE: A system for deciding quantified Boolean formulas satisfiability. *Proc. 1st Int. Conf. on Automated Reasoning* (IJCAR'01), R.Gor, A.Leitsch, and T.Nipkow, editors, LNCS 2083, Springer, pp.364–369, 2001.
19. S.Greco and E.Zumpano. Querying inconsistent databases. *Proc. Int. Conf. on Logic Programming and Automated Reasoning* (LPAR'2000), M.Parigot and A.Voronkov, editors, LNAI 1955, Springer, pp.308–325, 2000.
20. G.Greco, S.Greco, and E.Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. *Proc. 17th Int. Conf. on Logic Programming* (ICLP'01), LNCS 2237, Springer, pp.348–363, 2001.
21. H.Kleine-Büning, M.Karpinski, and A.Fögel. Resolution for quantified Boolean formulas. *Journal of Information and Computation* 177(1), pp.12–18, 1995.
22. R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. *Proc. TABLEAUX'2002*, U.Egly and G.C.Fermüler, editors, LNAI 2381, pp.160–175, 2002.

oarieli@mta.ac.il

23. P.Liberatore and M.Schaerf. BReLS: A system for the integration of knowledge bases. *Proc Int. Conf. on Principles of Knowledge Representation and Reasoning* (KR'2000), Morgan Kaufmann Publishers, pp.145–152, 2000.
24. J.McCarthy. Applications of circumscription to formalizing common-Sense knowledge. *Artificial Intelligence* 28, pp.89–116, 1986.
25. M.Moskewicz, C.Madigan, Y.Zhao, L.Zhang, and S.Malik. Chaff: Engineering an efficient SAT solver. *Proc. 39th Design Automation Conference*, 2001.
26. J.T.Rintanen. Improvements of the evaluation of quantified Boolean formulae. *Proc. 16th Int. Joint Conf. on Artificial Intelligence* (IJCAI'99), Morgan Kaufmann Publishers, pp.1192–1197. 1999.
27. C.Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science* 3(1), pp.23–33, 1976.

oarieli@mta.ac.il

Simplification of Integrity Constraints for Data Integration

Henning Christiansen and Davide Martinenghi

Roskilde University, Computer Science Dept.
P.O. Box 260, DK-4000 Roskilde, Denmark
{henning,dm}@ruc.dk

Abstract. When two or more databases are combined into a global one, integrity may be violated even when each database is consistent with its own local integrity constraints. Efficient methods for checking global integrity in data integration systems are called for: answers to queries can then be trusted, because either the global database is known to be consistent or suitable actions have been taken to provide consistent views. The present work generalizes simplification techniques for integrity checking in traditional databases to the combined case. Knowledge of local consistency is employed, perhaps together with given *a priori* constraints on the combination, so that only a minimal number of tuples needs to be considered. Combination from scratch, integration of a new source, and absorption of local updates are dealt with for both the local-as-view and global-as-view approaches to data integration.

1 Introduction

Data integration has attracted much attention in the recent years due to the explosion in online data sources and the whole aspect of globalization of society and business.

To integrate a set of different local data sources means to provide a common database schema, often called a global or mediator schema, and to describe a relationship between the different local schemata and the global one. Two common paradigms for defining such relationships are the so-called *local-as-view* (LaV, a.k.a. *source-centric*) and *global-as-view* (GaV, a.k.a. *global-centric*); see [28] for definitions and comparison.

Integrity constraints of a database are overall conditions that must be met by any instance of the database in order for it to provide a meaningful semantics, and maintenance of integrity constraints is a standard issue in traditional databases.

In combined databases, on the other hand, integrity constraints have been used mainly for query reformulation and optimization, but the problem of checking and maintaining integrity constraints in this context seems to be largely ignored (a few exceptions are mentioned in section 2). This is problematic as even though each local database may satisfy its specific integrity constraints, the combined database may not have a good semantics and the answers to queries cannot be trusted.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 31–48, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

Consider, as an example, two databases of marriages for two different countries. Both may satisfy a non-bigamist integrity constraint, but combining the two may violate this. We need methods to identify such violations and, ideally, provide means for restoring consistency, which may be done in different ways.

As in a traditional database of nontrivial size, it is not practically feasible to check integrity constraints for the entire database in one operation: an incremental approach is needed so that only a small amount of work is required for each update. For the combined case, it is even more urgent to optimize integrity checking and distribute it over time: Transition delays over network links need to be considered, and an update in this context may mean the addition of a new data source.

Theoretically sound methods, called *simplification* of integrity constraints, have been developed for relational and deductive databases, although the common practice is still based on *ad hoc* techniques. Typically, database experts need to design and hand-code either complicated tests in the program producing the update requests or triggers within the database management system that react upon certain update actions (and in combined databases, we expect it to be the same, if integrity is considered at all).

It seems obvious that a generalization of simplification techniques will be of great advantage for combined databases, and we can present some first results in adapting a newly developed and highly flexible simplification framework [13]. The main idea in simplification is to keep the database invariantly consistent, so that this knowledge can be utilized in the investigation whether the database will be consistent following a suggested update. For example, for a non-bigamist constraint, it means that it is sufficient to check the condition for new husbands and wives against the database when a marriage is proposed, as all other combinations of tuples have been checked earlier. Some simplification methods need to perform the update before the simplified constraint can be checked so that a roll-back operation may become necessary, whereas others such as [13] can do with the current state before the update, which seems to be a better approach. However, the method of [13] can also be adapted to perform post-update checks, which seems of relevance in a combined database where a local update is registered as a message that it has been performed.

With our approach we can check the integrity both when several sources are integrated and when an updated source notifies the mediator with a message about the update. In order to have a simplification, we need to have some knowledge about the current state. When integrating a new source, we may trust a statement that it satisfies its local constraints; when a message about a local update is received, we may trust two things: The combined database was consistent before the update and the update has been verified locally, i.e., the local constraints are maintained. As a natural result, we obtain that only the possible interference between the update and the other sources needs to be checked.

If global inconsistencies are found, we also indicate how integrity can be restored by maintaining at the global level a small database of virtual local changes, when it is not possible to modify the local sources directly.

oarieli@mta.ac.il

The paper is organized as follows. In section 2 we review existing literature in the field. The simplification framework of [13] is introduced in section 3 and its uses for data integration are explained in section 4. Examples of the GaV and LaV approaches are given in section 5 and 6 respectively, while the problem of dealing with updates at the source level is addressed in section 7. Concluding remarks and future directions are provided in section 8.

2 Related Work

The contribution of integrity constraints to data integration is usually confined to query reformulation and query optimization problems. In a GaV approach the global schema is expressed in terms of views over the sources, whereas in LaV the sources are formulated as views over the global schema. The former approach is usually considered simpler for query answering, as this typically amounts to unfolding a query with respect to the view definitions; on the other hand it is less flexible if new sources need to be added to (or removed from) the system. The latter has more involved query answer mechanisms, but enjoys good scalability, as changes at the sources do not require any modification in the global schema.

In [17] the problem of answering queries using views (*query folding*) under LaV is addressed with a technique based on resolution, and several cases, including integrity constraints, negation and recursion, are dealt with.

A foundational description of the problem of consistent query answering, without considering data integration issues, is given in [8], where analytic tableaux are used to characterize the repairs of databases that do not comply with given integrity constraints.

A short survey on the role of integrity constraints in data integration is given in [9]. They are regarded as means to extract more information from incomplete sources as well as components that raise the issue of dealing with possibly inconsistent global databases. Several GaV typologies are studied that include the treatment of key and foreign key constraints and both sound and exact mappings. In [10] the same authors develop these ideas to show that, in the presence of integrity constraints, query answering in GaV becomes as difficult as in LaV, as the problem of incomplete information implicitly arises. Further discussion on the expressive power of the two approaches, in terms of query-preserving transformations, is given in [11].

In [28], Ullman relates the problem of constructing answers to queries using views to query containment algorithms and compares two implemented systems in these terms.

Levy [20] applies techniques from artificial intelligence to the problem of data integration and shows examples of both GaV and LaV query reformulation with integrity constraints, including particular data access patterns.

Li [21] considers the use of integrity constraints in LaV query processing and optimization and distinguishes between local and global constraints and, for these, between general global constraints and source-derived global constraints.

oarieli@mta.ac.il

Others, e.g., [6,7,22], have approached the global consistency problem by introducing *disjunctive databases*, that we shall not consider in this paper. Conflicts in the data are usually resolved by means of majority principles; to this end, [26] extensively analyzes the general problem of arbitration, which is the process of settling a divergence, by considering preferences and weights. In addition to data-related inconsistencies, the authors of [22] also consider the problem of merging sources whose compatibility is affected by the presence of synonyms, homonyms or type conflicts in the schemata.

The presence of semi-structured data, i.e., information that does not match a strict predefined schema, is a common phenomenon in current applications, such as web databases, as also reflected in the emerging XML standard. This is an important issue in data integration and the database community has developed various techniques for handling this kind of information [1,5]; however, we will not focus on this problem in the remainder of the paper.

Another direction of research concerns automatic identification of a common global schema with mappings from different source schemata. Various techniques, such as linguistic and ontological similarity between relation and attribute names and type structures, can be used; see [27] for an overview. Recent work, e.g., [29], attempts to extract indirectly expressed similarities using a kind of shallow semantic analysis. Apart from referential integrity and key constraints, this research has not paid much attention to integrity constraints, but it seems very likely that a more fine-grained comparison of integrity constraints may help to identify semantic similarities.

Paraconsistent logics can be used to model the possible inconsistencies coming from database integration and update. In [3] it is shown how one such logic, called LFI1, has the capability of storing and managing inconsistent information.

What we want to do instead is to *check* the integrity of a database that consists of several data sources. In order to deal with this in an optimal way, in the next section we introduce new tools for the simplification of integrity constraints that can be used to manage a number of different configurations of database schemata and integrity constraints at the sources and at the mediator.

The principle of simplification of integrity constraints dates back to at least [24] and has been elaborated by many other authors. We develop this paper upon the framework described in [13] and we refer to that for further discussion, relevant proofs and references.

3 A Framework for Simplification of Integrity Constraints

We review here, with small variations, the simplification framework presented in [13] and assume a function-free first-order language equipped with negation and built-ins for equality ($\doteq$) and inequality ($\neq$), where *terms* $(t, s, \ldots)$, *variables* $(x, y, \ldots)$, *constants* $(a, b, \ldots)$, *predicates* $(p, q, \ldots)$, *atoms*, *literals* and *formulas* in general are defined as usual. The set of all ground (i.e., variable-free) atoms that can be formed from the predicate symbols and the terms in the language is referred to as the *Herbrand base*.

oarieli@mta.ac.il

Clauses are written in the form $Head \leftarrow Body$ where the head, if present, is an atom and the body a (perhaps empty) conjunction of literals. A *denial* is a headless clause and a *fact* is a bodiless clause; all other clauses are called *rules*. We identify a set of formulas with their conjunction and thus *true* with $\emptyset$ and ϕ with the set $\{\phi\}$. Logical equivalence between formulas is denoted by $\equiv$, entailment by $\models$. The notation $\vec{t}$ indicates a sequence of terms $t_1, \ldots, t_n$ and the expressions $p(\vec{t})$, $\vec{s} \doteq \vec{t}$ and $\vec{s} \neq \vec{t}$ are defined accordingly. We further assume that all clauses are *range restricted* (see, e.g., [13]). For simplicity, we do not allow recursion, but our method is relevant for all database environments in which range restricted queries produce a finite set of ground tuples.

We distinguish three components of a database: the *extensional database* (the facts), the *intensional database* (the rules) and the *constraint theory* (the integrity constraints) [16]. With no loss of generality [12], we shall assume throughout the rest of the paper that the constraint theory contains only extensional predicates. By *database state* we refer to the union of the extensional and the intensional parts only.

Definition 3.1 (Database). *A database is a triple $\langle S, \Gamma, D \rangle$, where Γ is a constraint theory, D a database state and S a set of signatures for the predicates of Γ and D, called a database signature.*

As semantics of a database state D, with default negation for negative literals, we take its *standard model*, as D is here recursion-free and thus *stratified*. The truth value of a closed formula F, relative to D, is defined as its valuation in the standard model and denoted $D(F)$. (See, e.g., [25] for exact definitions for these and other common logical notions.)

Definition 3.2 (Consistency). *A database state D is* consistent *with a constraint theory Γ iff $D(\Gamma) = true$.*

The method can handle general forms of update, including additions, deletions and changes. Furthermore, [13] allows also for so-called *parameters*, i.e., placeholders for constants that permit to generalize updates into update patterns, which can be evaluated before knowing the actual values of the update itself. For example, the notation $\{p(\mathbf{a}), \neg q(\mathbf{a})\}$, where $\mathbf{a}$ is a parameter, refers to the class of updates that add a tuple to the unary relation p and remove the same tuple from the unary relation q. For simplicity, in this paper we restrict our attention to parameter-free additions, but the results we present also hold in the presence of parameters as well as for deletions and changes.

Definition 3.3 (Update). *An* update *is a non-empty set of ground facts.*

3.1 Semantic Notions

We introduce now a few concepts that allow us to characterize the notion of simplification from a semantic point of view. We refer to [13] for further discussion.

The notion of weakest precondition is a semantic correctness criterion for a test to be run prior to the execution of the update, i.e., a test that can be checked in the present state but indicating properties of the prospective new state.

oarieli@mta.ac.il

Definition 3.4 (Weakest precondition, strongest postcondition). *Let Γ and Σ be constraint theories and U an update. Σ is a* weakest precondition *of Γ with respect to U and Γ is a* strongest postcondition *of Σ with respect to U whenever $D(\Sigma) \equiv (D \cup U)(\Gamma)$ for any database state D.*

Weakest preconditions [14,18] resemble the standard axiom for defining assignment statements in a programming language, whose side effects are analogous to a database update. The concept of strongest postcondition characterizes how questions concerning the previous state may be answered in the updated state.

The essence of simplification is the optimization of a weakest precondition based on the invariant that the constraint theory holds in the present state.

Definition 3.5 (Conditional weakest precondition). *Let Γ, Δ be constraint theories and U an update. A constraint theory Σ is a* Δ-conditional weakest precondition *(Δ-CWP) of Γ with respect to U whenever $D(\Sigma) \equiv (D \cup U)(\Gamma)$ for any database state D consistent with Δ.*

Typically, Δ will include Γ, but it may also contain other knowledge, such as further properties of the database that are trusted.

The notion of CWP alone is not sufficient to fully characterize the principle of simplification: an optimality criterion is needed, which serves as an abstraction over actual computation times without introducing assumptions about any particular evaluation mechanism or referring to any specific database state. According to definition 3.5, semantically different Δ-CWPs may exist, as their truth values are only fixed in the states that are consistent with Δ. Among these, we characterize as optimal a constraint theory that depends on as small a part of the database as possible (see definition 3.8). The following example demonstrates this idea and shows that a semantically weakest Δ-CWP, i.e., one that holds in as many states as possible, does not, in general, enjoy this property.

Example 3.1. Consider the constraint theory $\Gamma = \Delta = \{\leftarrow p(a) \wedge q(a),\ \leftarrow r(a)\}$ and the update $U = \{p(a)\}$. The strongest, optimal and weakest Δ-CWPs of Γ with respect to U are shown in the following table.

Strongest	Optimal	Weakest
$\{\leftarrow q(a),\ \leftarrow r(a)\}$	$\{\leftarrow q(a)\}$	$\{\leftarrow q(a) \wedge \neg p(a) \wedge \neg r(a)\}$

We base our definition of optimality on the notion of cover, i.e., a portion of the Herbrand base that does not affect the semantics of a constraint theory: the larger the cover, the better the constraint theory.

Definition 3.6 (Cover). *Two database states D_1, D_2 are said to* agree *on a subset $\mathcal{S}$ of the Herbrand base $\mathcal{B}$ whenever $D_1(A) = D_2(A)$ for every ground atom $A \in \mathcal{S}$. A set $\mathcal{C} \subseteq \mathcal{B}$ is a* cover *for a constraint theory Γ whenever*

$$D(\Gamma) = D'(\Gamma)$$

for any two database states D, D' that agree on $\mathcal{B} \setminus \mathcal{C}$. If, furthermore, Γ admits no other cover $\mathcal{C}' \supset \mathcal{C}$, $\mathcal{C}$ is a maximal cover *for Γ.*

oarieli@mta.ac.il

Definition 3.7 (Conditional equivalence). *Let Δ, Γ_1, Γ_2 be constraint theories, then Γ_1 and Γ_2 are conditionally equivalent with respect to Δ, denoted $\Gamma_1 \overset{\Delta}{\equiv} \Gamma_2$, whenever $D(\Gamma_1) \equiv D(\Gamma_2)$ for any database state D consistent with Δ.*

Definition 3.8 (Optimality). *Given two constraint theories Δ and Σ, Σ is* Δ-optimal *if it admits a maximal cover $\mathcal{C}$ such that there exists no other constraint theory $\Sigma' \overset{\Delta}{\equiv} \Sigma$ with maximal cover $\mathcal{C}' \supset \mathcal{C}$. A Δ-optimal Δ-CWP of a constraint theory Γ with respect to an update U is a* Δ-optimal conditional weakest precondition *(Δ-OCWP) of Γ with respect to U.*

Obviously the integrity constraint $\{\leftarrow q(a)\}$ indicated as optimal in example 3.1 satisfies this definition, as it admits the maximal cover $\mathcal{B} \setminus \{q(a)\}$. The only larger cover is $\mathcal{B}$ and any constraint theory with cover $\mathcal{B}$ would not be a Δ-CWP of Γ with respect to U in the example.

Ideally, the test for possible inconsistency introduced by a given update should be performed prior to the update, but in the setting of data integration this might not always be feasible. A consistency test that can be made after the update is called a weakest post-precondition. Similarly to Δ-CWPs, we refer to the following conditional notion.

Definition 3.9 (Conditional weakest post-precondition). *Let Γ, Δ be constraint theories and U an update. A constraint theory Σ is a* Δ-conditional weakest post-precondition *(Δ-CWPP) of Γ with respect to U whenever $(D \cup U)(\Sigma) \equiv (D \cup U)(\Gamma)$ for any database state D consistent with Δ.*

The notion of optimal Δ-CWPP can be defined analogously to definition 3.8 and is indicated as Δ-OCWPP.

3.2 Transformations on Integrity Constraints

In the following, we define the transformations that are used to compose a simplification procedure.

Definition 3.10. *Let Γ be a constraint theory and U an update:*

$$\begin{aligned} U = \{\ & p_1(\vec{a}_{1,1}), p_1(\vec{a}_{1,2}), \ldots, p_1(\vec{a}_{1,n_1}), \\ & \ldots \\ & p_m(\vec{a}_{m,1}), p_m(\vec{a}_{m,2}), \ldots, p_m(\vec{a}_{m,n_m}))\}, \end{aligned}$$

where the p_i's are distinct predicates and the $\vec{a}_{i,j}$'s are sequences of constants. The notation $\mathsf{After}^U(\Gamma)$ refers to a copy of Γ in which all atoms of the form $p_i(\vec{t})$ have been simultaneously replaced by

$$p_i(\vec{t}) \vee \vec{t} \doteq \vec{a}_{i,1} \vee \cdots \vee \vec{t} \doteq \vec{a}_{i,n_i}.$$

The notation $\mathsf{Before}^U(\Gamma)$ is as $\mathsf{After}^U(\Gamma)$ but where the replacement is

$$p_i(\vec{t}) \wedge \vec{t} \neq \vec{a}_{i,1} \wedge \cdots \wedge \vec{t} \neq \vec{a}_{i,n_i}.$$

oarieli@mta.ac.il

It follows immediately from the definition that After and Before distribute over $\cup$. We furthermore assume that the result of these transformations is always given as a set of denials, obtained by applications of De Morgan's laws, and in a "normalized" form, i.e., without redundant constraints and sub-formulas (such as, e.g., $a \doteq a$). We refer to [13] for a proposed implementation. The semantic correctness of After and Before is expressed by the following property.

Theorem 3.1. *For any update U and constraint theory Γ, $\mathsf{After}^U(\Gamma)$ is a weakest precondition and $\mathsf{Before}^U(\Gamma)$ is a strongest postcondition of Γ with respect to U.*

An essential step in the achievement of simpler integrity constraints is to employ the fact that they hold in the current database state, and remove those parts of the condition about the possible updated state that are implied by this. For this purpose, we introduce a transformation Optimize that produces a constraint theory which is optimal with respect to a given set of hypotheses.

Definition 3.11 (Optimize). *Given two constraint theories Δ, Γ, $\mathsf{Optimize}_\Delta(\Gamma)$ refers to a Δ-optimal theory Σ such that $\Sigma \overset{\Delta}{\equiv} \Gamma$.*

It should be observed that Optimize is defined in a purely mathematical way that does not indicate how to construct such a constraint theory. In the following, we shall refer to the implementation given in [13], based on the purely syntactic notion of subsumption (see, e.g., [16]): Optimize removes from Γ all denials that are subsumed by a denial in Δ. Although we have no proof for it yet, we believe that this implementation produces the desired results, at least under reasonable conditions for Γ. From definition 3.11 we have immediately the following.

Proposition 3.1. *Let Σ be a weakest precondition of constraint theory Γ with respect to an update U. Then $\mathsf{Optimize}_\Delta(\Sigma)$, Δ a constraint theory, is a Δ-OCWP of Γ with respect to U.*

The operators introduced so far can be combined to define a procedure for simplification of integrity constraints, where the updates always take place from a consistent state.

Definition 3.12. *For a constraint theory Γ and an update U, we define*

$$\mathsf{Simp}^U(\Gamma) = \mathsf{Optimize}_\Gamma(\mathsf{After}^U(\Gamma)).$$

As a consequence of the previous results, Simp enjoys the following property.

Proposition 3.2. *Let Γ be a constraint theory and U an update. Then $\mathsf{Simp}^U(\Gamma)$ is a Γ-OCWP of Γ with respect to U.*

No other work we are aware of has based simplification on the notion of optimal conditional weakest precondition.

oarieli@mta.ac.il

Example 3.2. Consider a database containing information about marriages, which receives updates of the form $U = \{m(a, b)\}$ (husband a is married to wife b) and has the following integrity constraint (no husband has more than a wife):

$$\Gamma = \leftarrow m(x, y) \wedge m(x, z) \wedge y \neq z.$$

The simplification, showing intermediate steps in After, is calculated as follows.

$$\begin{aligned}
\mathsf{After}^U(\Gamma) \equiv \{ & \leftarrow m(x, y) \wedge m(x, z) \wedge y \neq z, \\
& \leftarrow m(x, y) \wedge x \doteq a \wedge z \doteq b \wedge y \neq z, \\
& \leftarrow x \doteq a \wedge y \doteq b \wedge m(x, z) \wedge y \neq z, \\
& \leftarrow x \doteq a \wedge y \doteq b \wedge x \doteq a \wedge z \doteq b \wedge y \neq z \} \equiv \\
\equiv \mathsf{After}^U(\Gamma) = \{ & \leftarrow m(x, y) \wedge m(x, z) \wedge y \neq z, \\
& \leftarrow m(a, y) \wedge y \neq b\}
\end{aligned}$$

$$\mathsf{Simp}^U(\Gamma) = \{\leftarrow m(a, y) \wedge y \neq b\}$$

There may be specific applications where consistency needs to be checked directly on the updated database. In these cases, Before comes in handy.

Definition 3.13. *For a constraint theory Γ and an update U, we define*

$$\mathsf{PostSimp}^U(\Gamma) = \mathsf{Before}^U(\mathsf{Simp}^U(\Gamma)).$$

Proposition 3.3. *Let D be a database state consistent with a constraint theory Γ and U an update. Then $(D \cup U)(\Gamma) \equiv (D \cup U)(\mathsf{PostSimp}^U(\Gamma))$.*

Proposition 3.4. *Let Γ be a constraint theory and U an update. Then $\mathsf{PostSimp}^U(\Gamma)$ is a Γ-OCWPP of Γ with respect to U.*

4 Integrity Constraints in Combined Databases

4.1 Extending the Framework for Data Integration

In section 3 we described a framework that applies to a single updatable database. In the context of data integration, we have in general several databases (the sources and the mediator) and other operations than database updates need to be considered, such as database combination. We shall therefore generalize the notation in order to describe the relevant cases for data integration. In order to provide a unified view of the data residing at different sources, we need to indicate how the global predicates are expressed in terms of the source predicates. We shall therefore introduce the notion of mapping, which we assume to be *sound*, i.e., the information produced by the views over the sources contains only, but not necessarily all the data associated to the global predicates. *Complete* mappings and *exact* mappings are defined in a similar way (see, e.g., [10]), but we do not consider them in this paper.

oarieli@mta.ac.il

Definition 4.1 (Mapping). *A mapping* $M : (\mathcal{D}_1, ..., \mathcal{D}_n) \to \mathcal{D}$*, where* $\mathcal{D}$*,* $\mathcal{D}_1$*, ...,* $\mathcal{D}_n$ *are databases with disjoint signatures, is a set of range restricted rules in which the predicates in the heads are in* $\mathcal{D}$ *and their terms are distinct variables, and the predicates in the bodies are in one of the* $\mathcal{D}_1, ..., \mathcal{D}_n$ *or are built-ins.*

Notice that it is always possible to rewrite a set of range restricted rules as a mapping: for a rule whose head contains some constants or non distinct variables, they can be replaced by new variables, provided that equalities taking track of the replacements are added in the body.

We can now extend the After operator to handle data integrations described by a mapping from the sources to the mediator.

Definition 4.2. *Let* $\mathcal{D}, \mathcal{D}_1, ..., \mathcal{D}_n$ *be some databases with disjoint signatures,* Γ *a constraint theory concerning the predicates in* $\mathcal{D}$ *and* M *a mapping of the form:*

$$M = \{\ p_1(\vec{x}_1) \leftarrow B_{1,1}, \quad \ldots,\ p_1(\vec{x}_1) \leftarrow B_{1,n_1},$$
$$\vdots$$
$$p_m(\vec{x}_m) \leftarrow B_{m,1},\ \ldots,\ p_m(\vec{x}_m) \leftarrow B_{m,n_m}\},$$

where the p_i*'s are all the (distinct) predicates in* $\mathcal{D}$*, the* $\vec{x}_i$*'s are sequences of variables and the* $B_{i,j}$*'s are conjunctions of literals whose predicates are in one of the* $\mathcal{D}_1, ..., \mathcal{D}_n$ *or are built-ins. The notation* $\mathsf{After}^M(\Gamma)$*, where* $M : (\mathcal{D}_1, ..., \mathcal{D}_n) \to \mathcal{D}$ *is a mapping, refers to a copy of* Γ *in which all atoms of the form* $p_i(\vec{t})$ *have been simultaneously replaced by*

$$B_{i,1}\theta_i\rho_{i,1} \vee ... \vee B_{i,n_i}\theta_i\rho_{i,n_i},$$

where θ_i *is a substitution that replaces the variables of* $\vec{x}_i$ *with the terms of* $\vec{t}$ *and* $\rho_{i,j}$ *a renaming giving fresh new names to the variables of* $B_{i,j}$ *not in* $\vec{x}_i$ *to avoid name clashes.*

We use the term *operation* to refer to either an update or the application of a mapping. The After operator, as stated by definitions 3.10 and 4.2, behaves as follows: for an update, it translates a theory concerning the updated state to one referring to the state before the update; similarly, for a database combination, it moves from the state after the integration to the non-integrated state, i.e., from a theory concerning the mediator to one concerning the sources. It is therefore meaningful to extend the notion of weakest precondition accordingly.

Definition 4.3. *Let* Γ *and* Σ *be constraint theories and* $M : (\mathcal{D}_1, ..., \mathcal{D}_n) \to \mathcal{D}$ *a mapping where* $\mathcal{D}, \mathcal{D}_1, ..., \mathcal{D}_n$ *are databases.* Σ *is a* weakest precondition *of* Γ *with respect to* M *whenever* $(D_1 \cup ... \cup D_n)(\Sigma) \equiv D(\Gamma)$ *for any database states* $D, D_1, ..., D_n$ *in* $\mathcal{D}, \mathcal{D}_1, ..., \mathcal{D}_n$*.*

The notions of CWP and OCWP can be extended in a similar way. We immediately have the extension of theorem 3.1 and proposition 3.1, where the word "operation" can be used instead of "update".

In definition 3.12 we implicitly assumed that the argument of Simp was both the theory to be simplified and the condition known to hold prior to the operation; it is now useful to extend the notation as follows.

oarieli@mta.ac.il

Definition 4.4. *For two constraint theories Γ, Δ and an operation O, we define*

$$\mathsf{Simp}^O_\Delta(\Gamma) = \mathsf{Optimize}_\Delta(\mathsf{After}^O(\Gamma)).$$

Proposition 3.2 can immediately be generalized as follows.

Proposition 4.1. *Let Γ, Δ be constraint theories and O an operation. Then $\mathsf{Simp}^O_\Delta(\Gamma)$ is a Δ-OCWP of Γ with respect to O.*

The notation $\mathsf{Simp}^U(\Gamma)$ of definition 3.12, U an update, can therefore be considered as a shorthand for $\mathsf{Simp}^U_\Gamma(\Gamma)$. Examples of the application of these extended versions of the operators are shown in the next sections.

4.2 Consistent Views on Inconsistent States

When the information contained in the source databases is conflicting with the global requirements and the sources cannot be modified, it is possible to repair inconsistencies by maintaining a virtual database of exceptions $\mathcal{D}_E$ at the global level. Suppose sources $\mathcal{D}_1, ..., \mathcal{D}_n$ are given and the global database $\mathcal{D}$ is defined via a mapping $M : (\mathcal{D}_1, ..., \mathcal{D}_n) \to \mathcal{D}$. Let $\mathcal{D}_E$'s signature contain for each predicate p in $\mathcal{D}$ two predicates p_+ and p_- of the same arity, the former representing the tuples that should be in p but are not, the latter those that should not be in p but are. The consistent global database $\mathcal{D}_C$ can then be thought of as a combination of $\mathcal{D}_E$ and $\mathcal{D}$ that contains a predicate p_C for every predicate p in $\mathcal{D}$, as defined in the mapping $M_E : (\mathcal{D}, \mathcal{D}_E) \to \mathcal{D}_C$ by the following entries:

$$p_C(\vec{x}) \leftarrow p(\vec{x}) \wedge \neg p_-(\vec{x}),$$
$$p_C(\vec{x}) \leftarrow p_+(\vec{x}).$$

In order to identify suitable p_+ and p_- to re-establish consistency, an abductive algorithm can be integrated with the procedure for evaluating the simplified integrity constraints. A full treatment of this topic is outside the scope of this paper; see, e.g., [15,19] for an overview of abductive methods. Applications of abduction to database repair that fit our model are described in [2,4].

5 Integrity Constraints under Global-as-View

In a global-centric approach it is required that the global schema is expressed in terms of the sources. The global predicates must be associated with views over the sources: this is exactly what the notion of mapping makes precise, as stated in definition 4.1. A mapping containing entries for all global predicates is called a GaV-mapping.

We start our analysis by considering a borderline case of GaV-mapping consisting of the combination of two[1] databases having the same signature and constraint theory. Let S_1, S_2, S be three disjoint database signatures that are

[1] The case with more than two sources is similar.

oarieli@mta.ac.il

identical up to consistent renaming of predicates and Γ_1, Γ_2 and Γ the constraint theories (also identical up to renaming) defined at the sources and the mediator, respectively. The global database state consists of the union of the local ones and the mapping M used for the combination is defined as a set containing, for every predicate p in S, the entries:

$$p(\vec{x}) \leftarrow p_1(\vec{x}),$$
$$p(\vec{x}) \leftarrow p_2(\vec{x}),$$

where $\vec{x}$ is a sequence of variables and p_1, p_2 are the predicates corresponding to p in S_1, S_2, respectively. A simplified test for checking that the combined database is consistent with Γ is then given by $\mathsf{Simp}^M_{\Gamma_1 \wedge \Gamma_2}(\Gamma)$.

Example 5.1. Let us refer to example 3.2 and consider two sources $\mathcal{D}_1 = \langle S_1, \Gamma_1, D_1 \rangle$, $\mathcal{D}_2 = \langle S_2, \Gamma_2, D_2 \rangle$ and a mediator $\mathcal{D} = \langle S, \Gamma, D \rangle$ with signatures[2] $S_1 = \{m_1/2\}$, $S_2 = \{m_2/2\}$, $S = \{m/2\}$ and the following integrity constraints:

$$\begin{aligned} \Gamma &= \leftarrow m(x,y) \wedge m(x,z) \wedge y \neq z, \\ \Gamma_1 &= \leftarrow m_1(x,y) \wedge m_1(x,z) \wedge y \neq z, \\ \Gamma_2 &= \leftarrow m_2(x,y) \wedge m_2(x,z) \wedge y \neq z. \end{aligned}$$

D_1, D_2 are consistent database states and D is their combination, as expressed by the mapping: $M = \{m(x,y) \leftarrow m_1(x,y),\ m(x,y) \leftarrow m_2(x,y)\}$. We have:

$$\begin{aligned} \mathsf{After}^M(\Gamma) \equiv \{ \leftarrow\ & (m_1(x,y) \vee m_2(x,y)) \wedge \\ & (m_1(x,z) \vee m_2(x,z)) \wedge y \neq z \ \} \equiv \\ \equiv \mathsf{After}^M(\Gamma) = \{ \leftarrow\ & m_1(x,y) \wedge m_1(x,z) \wedge y \neq z, \\ \leftarrow\ & m_1(x,y) \wedge m_2(x,z) \wedge y \neq z, \\ \leftarrow\ & m_2(x,y) \wedge m_2(x,z) \wedge y \neq z\} \end{aligned}$$

The only check that is needed is, as expected, a cross-check between the two databases, as the other denials are removed by $\mathsf{Optimize}$:

$$\mathsf{Simp}^M_{\Gamma_1 \wedge \Gamma_2}(\Gamma) = \{ \leftarrow m_1(x,y) \wedge m_2(x,z) \wedge y \neq z \ \}.$$

We can get even better results when extra knowledge concerning the combination of the sources is available. This simply amounts to adding the extra knowledge to the conditions in the subscript of Simp.

Example 5.2. Consider example 5.1, where we have now the knowledge $\Gamma_{1,2}$ that the data concerning the husbands in the two databases are disjoint:

$$\Gamma_{1,2} = \leftarrow m_1(x,y) \wedge m_2(x,z).$$

A much stronger simplification is obtained now, as

$$\mathsf{Simp}^M_{\Gamma_1 \wedge \Gamma_2 \wedge \Gamma_{1,2}}(\Gamma) = \mathit{true}.$$

The cross-check that was found in example 5.1 is subsumed by $\Gamma_{1,2}$ and thus discarded, so no check is needed, as the combined state will anyhow be consistent.

[2] As usual, predicate signatures are indicated as $name/arity$.

oarieli@mta.ac.il

When the mapping is arbitrary, the method can be applied in a similar way.

Example 5.3. We consider a data integration problem inspired by Levy [20] but here extended with global and local integrity constraints. Suppose we have two sources containing information about movies. We use the variables i, t, y and r for movie identifiers, titles, years and reviews respectively. The first source contains movies $m(i,t,y)$, where i is key, whereas the second contains reviews $r(i,r)$. Furthermore, we know that the identifiers in r are a subset of the identifiers in m. The mediator assembles this information in a relation $f(i,t,r)$ (film), as defined by the following GaV-mapping:

$$M = \{f(i,t,r) \leftarrow m(i,t,y) \wedge r(i,r)\}.$$

The following conditions are therefore known to hold on the ensemble of sources:

$$\begin{aligned} \Gamma_1 = \{ & \leftarrow m(i,t_1,y_1) \wedge m(i,t_2,y_2) \wedge t_1 \neq t_2, \\ & \leftarrow m(i,t_1,y_1) \wedge m(i,t_2,y_2) \wedge y_1 \neq y_2\}, \\ \Gamma_{1,2} = \quad & \leftarrow r(i,r) \wedge \neg m(i,t,y). \end{aligned}$$

Let Γ express the fact that i is a primary key for f:

$$\begin{aligned} \Gamma = \{ & \leftarrow f(i,t_1,r_1) \wedge f(i,t_2,r_2) \wedge t_1 \neq t_2, \\ & \leftarrow f(i,t_1,r_1) \wedge f(i,t_2,r_2) \wedge r_1 \neq r_2\}. \end{aligned}$$

In order to check whether Γ holds globally, we proceed as follows.

$$\begin{aligned} \mathsf{After}^M(\Gamma) = \{ & \leftarrow m(i,t_1,y_1) \wedge r(i,r_1) \wedge m(i,t_2,y_2) \wedge r(i,r_2) \wedge t_1 \neq t_2, \\ & \leftarrow m(i,t_1,y_1) \wedge r(i,r_1) \wedge m(i,t_2,y_2) \wedge r(i,r_2) \wedge r_1 \neq r_2\}. \end{aligned}$$

The first constraint is obviously subsumed by the first constraint in Γ_1 and the second one can be simplified with $\Gamma_{1,2}$, which gives the following:

$$\mathsf{Simp}^M_{\Gamma_1 \wedge \Gamma_{1,2}}(\Gamma) = \{\leftarrow r(i,r_1) \wedge r(i,r_2) \wedge r_1 \neq r_2\},$$

which evidently corresponds to the fact that i must be a key for r as well.

6 Integrity Constraints under Local-as-View

A LaV-mapping is usually understood as a set of views of source predicates over global predicates. From now on with the word LaV-view we will refer to a formula of the form $A \rightarrow B$, where A is an atom (the antecedent), B a conjunction of atoms (the consequents) and universal quantification at the outmost level is understood for all variables. A LaV-view $A \rightarrow B$ is *safe* whenever all the variables in B appear in A as well. A set of safe LaV-views is called a safe LaV-mapping. Given a safe[3] LaV-mapping, and assuming it is sound as discussed in section 4, we can always rewrite it as an equivalent mapping. This is shown in

[3] Safeness is used to avoid skolemization.

oarieli@mta.ac.il

the following examples and can be done by using the fact that, with safeness, whenever $A \rightarrow A_1 \wedge ... \wedge A_n$, then $A_1 \leftarrow A$ and ... and $A_n \leftarrow A$, where $A, A_1, ..., A_n$ are atoms, and perhaps by adding some equalities in the bodies in order to have only distinct variables in the heads. Note that in general a LaV-mapping is not necessarily a mapping in the sense of definition 4.1.

Example 6.1. Reconsider the scenario discussed in example 5.1. The global database combines now two sources where in the first one the husbands are Italian, and in the second one Danish, which is expressed by the LaV-mapping $L = \{m_1(x,y) \rightarrow m(x,y) \wedge n(x,it),\ m_2(x,y) \rightarrow m(x,y) \wedge n(x,dk)\}$, where m and n are global predicates. An equivalent mapping is as follows:

$$\begin{aligned} M_L = \{\ & m(x,y) \leftarrow m_1(x,y), \\ & n(x,z) \leftarrow m_1(x,y) \wedge z \doteq it, \\ & m(x,y) \leftarrow m_2(x,y), \\ & n(x,z) \leftarrow m_2(x,y) \wedge z \doteq dk\}. \end{aligned}$$

We note that, given the local no-bigamist assumptions Γ_1 and Γ_2, the simplification of uniqueness of nationality $\mathsf{Simp}^{M_L}_{\Gamma_1 \wedge \Gamma_2}(\leftarrow n(x,y) \wedge n(x,z) \wedge y \neq z) = \leftarrow m_1(x,y) \wedge m_2(x,z)$ corresponds to the disjointness of m_1 and m_2.

Example 6.2. This example is also inspired by Levy ([20]) and extended with global and local integrity constraints. The global database integrates three different sources that provide information about movies. We use the variables t, y, d and g for movie titles, years, directors, and genres respectively. The global predicates are $m(t,y,d,g)$, representing a given movie, and $d(d)$, $i(d)$, $a(d)$, ..., representing nationalities of directors, here Danish, Italian, American, etc. The following integrity constraints are assumed: a key constraint on m (t,y is key), a domain constraint on film genre, and uniqueness of nationality. Underlines are used as anonymous variables *à la* Prolog for ease of notation.

$$\begin{aligned} \Gamma = \{\ & \leftarrow m(t,y,d_1,\_) \wedge m(t,y,d_2,\_) \wedge d_1 \neq d_2, \\ & \leftarrow m(t,y,\_,g_1) \wedge m(t,y,\_,g_2) \wedge g_1 \neq g_2, \\ & \leftarrow m(\_,\_,\_,g) \wedge g \neq \mathit{comedy} \wedge g \neq \mathit{drama} \wedge \cdots, \\ & \leftarrow d(d) \wedge i(d), \\ & \leftarrow d(d) \wedge a(d), \\ & \cdots\}. \end{aligned}$$

There are three source databases. The first one contains American comedies given as $m_1(t,y,d)$ with t,y as key and a LaV-mapping as follows.

$$\begin{aligned} \Gamma_1 &= \leftarrow m_1(t,y,d_1) \wedge m_1(t,y,d_2) \wedge d_1 \neq d_2 \\ L_1 &= \{m_1(t,y,d) \rightarrow m(t,y,d,\mathit{comedy}) \wedge a(d)\}. \end{aligned}$$

The second source contains Danish movies only, with a key constraint Γ_2 on t,y as usual and the following LaV-view.

$$L_2 = \{m_2(t,y,d,g) \rightarrow m(t,y,d,g) \wedge d(d)\}.$$

oarieli@mta.ac.il

The third source is a general list of movies with predicates similar to the global ones (m_3, d_3, i_3, a_3 etc.) and with similar integrity constraints Γ_3. The LaV-views are specified as follows.

$$\begin{aligned} L_3 = \{\ & m_3(t,y,d,g) \rightarrow m(t,y,d,g), \\ & d_3(d) \rightarrow d(d), \\ & \cdots\} \end{aligned}$$

As $L_1 \cup L_2 \cup L_3$ is safe we can rewrite it as the following mapping.[4]

$$\begin{aligned} M = \{\ & m(t,y,d,g) \leftarrow m_1(t,y,d) \wedge g \doteq \mathit{comedy}, \\ & m(t,y,d,g) \leftarrow m_2(t,y,d,g), \\ & m(t,y,d,g) \leftarrow m_3(t,y,d,g), \\ & a(d) \leftarrow m_1(\_,\_,d), \\ & d(d) \leftarrow m_2(\_,\_,d,\_), \\ & a(d) \leftarrow a_3(d), \\ & d(d) \leftarrow d_3(d), \\ & i(d) \leftarrow i_3(d), \\ & \cdots\} \end{aligned}$$

The simplified integrity constraints for the integration of the three databases, given as $\Sigma = \mathsf{Simp}^{M}_{\Gamma_1 \wedge \Gamma_2 \wedge \Gamma_3}(\Gamma)$, are, as expected, simplified rules covering possible conflicts in cross combinations only, as local consistency is assumed.

$$\begin{aligned} \Sigma = \{\ & \leftarrow m_1(\_,\_,d) \wedge m_2(\_,\_,d,\_), \\ & \leftarrow m_1(\_,\_,d) \wedge \mathbf{p}_3(d), \quad (\mathbf{p} \text{ any nationality pred. different from } a) \\ & \leftarrow m_2(\_,\_,d,\_) \wedge \mathbf{p}_3(d), \quad (\mathbf{p} \text{ any nationality pred. different from } d) \\ & \leftarrow m_1(t,y,\_) \wedge m_2(t,y,\_,\_), \\ & \leftarrow m_1(t,y,d_1) \wedge m_3(t,y,d_2,\_) \wedge d_1 \neq d_2, \\ & \leftarrow m_2(t,y,d_1,\_) \wedge m_3(t,y,d_2,\_) \wedge d_1 \neq d_2, \\ & \leftarrow m_1(t,y,\_) \wedge m_3(t,y,\_,g) \wedge g \neq \mathit{comedy}, \\ & \leftarrow m_2(t,y,\_,g_1) \wedge m_3(t,y,\_,g_2) \wedge g_1 \neq g_2, \\ & \leftarrow m_2(\_,\_,\_,g) \wedge g \neq \mathit{comedy} \wedge g \neq \mathit{drama} \wedge \cdots\}. \end{aligned}$$

The example can be changed a bit assuming that the third source is an unchecked database to which enthusiastic amateurs can add arbitrary information, which is not unrealistic in case of the world wide web. In that case, the simplified constraints include also a copy of the full set of global constraints with predicate names m_3, a_3, etc.

7 Absorption of Local Updates

A data integration system needs to be able to adjust itself dynamically as sources are updated over time. We assume that source databases maintain their own consistency and that reports about which updates have been performed are

[4] If, say, in m_1 the genre was left unspecified, skolemization would be needed.

oarieli@mta.ac.il

available at the global level; this may be supplied by the source administrator or generated by a process monitoring the sources. Consistency needs then to be checked globally. This problem, that we refer to as *absorption* of local updates, can be handled in an optimal way as described in the following proposition.

Proposition 7.1. *Given n sources with database states D_i, $1 \leq i \leq n$, a mediator $\langle S, \Gamma, D\rangle$ obtained with mapping M and a set of conditions Δ holding in $D_S = D_1 \cup ... \cup D_n$, let $\Sigma = \mathsf{Simp}^M_\Delta(\Gamma)$ hold in D, U be an update for D_S and D^U the state at the mediator after the update. Then $D^U(\Gamma) \equiv D_S(\mathsf{PostSimp}^U(\Sigma))$.*

Note that the condition expressed by $\mathsf{PostSimp}^U(\Sigma)$ is optimal in the sense of proposition 3.4.

Example 7.1. In example 5.1, the following integrity constraint was generated for the integration of two sources referred to by predicates m_1 and m_2:

$$\Sigma = \leftarrow m_1(x, y) \wedge m_2(x, z) \wedge y \neq z.$$

If the update $m_1(a, b)$ is reported, the optimal way to check the global consistency is to test $\mathsf{PostSimp}^U(\Sigma) = \leftarrow m_2(a, z) \wedge b \neq z$ on the updated state.

Example 7.2. Consider Σ from the LaV integrated movie database of example 6.2 and assume that the second source reports the addition of a new movie

$$U = \{m_2(\textit{“Dogville”}, 2003, \textit{“von Trier”}, \textit{drama})\}.$$

The following tests, calculated as $\mathsf{PostSimp}^U(\Sigma)$, remain:

$$\begin{array}{l}\{ \leftarrow m_1(\_, \_, \textit{“von Trier”}),\\ \quad \leftarrow \mathbf{p}_3(\textit{“von Trier”}), \quad (\mathbf{p} \text{ any nationality pred. different from } d)\\ \quad \leftarrow m_1(\textit{“Dogville”}, 2003, \_),\\ \quad \leftarrow m_3(\textit{“Dogville”}, 2003, d, \_) \wedge d \neq \textit{“von Trier”},\\ \quad \leftarrow m_3(\textit{“Dogville”}, 2003, \_, g) \wedge g \neq \textit{drama}\}.\end{array}$$

8 Conclusion

We revisited simplification techniques for integrity constraints and applied them to the problem of consistency checking in a data integration setting. Examples were discussed that showed that the described method lends itself well to both the LaV and GaV approaches and is useful for the problem of update absorption.

The novelty of this approach mainly consists of two aspects: firstly, the reuse of a simplification framework that was initially conceived for a single database and, secondly, the characterization of the semantic notions involved in the combined case that underlie the optimality of the method.

The treatment of cases that require skolemization when converting LaV-mappings to mappings has not been attempted in this paper. Future directions include the investigation of new strategies, if needed, that deal with such cases as well as the extension of the method to situations where the mapping is not sound, but exact or complete.

oarieli@mta.ac.il

Acknowledgements. The authors wish to thank Amos Scisci for his comments on the first draft of the manuscript. This research is supported in part by the IT-University of Copenhagen.

References

1. Abiteboul, S. (1997). Querying semi-structured data. *Proc. of the Int. Conf on Database Theory (ICDT)*, pp. 1–18, Vol. 1186, Springer LNCS.
2. Arenas, M., Bertossi, L., Chomicki, J., (2000). Specifying and Querying Database Repairs using Logic Programs with Exceptions. *Proceedings of FQAS 2000*, Larsen, H., Kacprzyk, J. Zadrozny, S., Andreasen, T., Christiansen, H. (eds.), pp. 27–41, Physica-Verlag Heidelberg New York, A Springer-Verlag Company.
3. De Amo, S., Carnielli, W., Marcos, J. (2002). A Logical Framework for Integrating Inconsistent Information in Multiple Databases. *FoIKS 2002*, Proceedings, Eds. Eiter, T. and Schewepp, K., pp. 67–84, Vol. 2284, Springer LNCS.
4. Arieli, O., Denecker, M., Van Nuffelen, B., Bruynooghe, M. (2002). Repairing Inconsistent Databases: A Model-Theoretic Approach and Abductive Reasoning. *PCL 2002*, Decker, H., Villadsen, J., Waragaipp, T. pp. 51–65, Datalogiske Skrifter, Vol. 95, Roskilde University, Roskilde, Denmark.
5. Buneman, P. (1997). Semistructured data. *Proc. of the ACM SIGACT-SIGMOD-SIGART Smposium on Principles of Database Systems (PODS)*, pp- 117–121, ACM Press.
6. Baral, C., Kraus, S., Minker, J., (1991). Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*. 3(2), pp. 208–220.
7. Baral, C., Kraus, S., Minker, J., Subrahmanian, S. (1992). Combining knowledge bases consisting of first-order theories. *Computational Intelligence*, 8, pp. 45–71.
8. Bertossi, L., Schwind, C. (2002). Analytic Tableaux and Database Repairs: Foundations.*FoIKS 2002*, Proceedings, Eds. Eiter, T. and Schewepp, K., pp. 32–48, Vol. 2284, Springer LNCS.
9. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M. (2002). On the Role of Integrity Constraints in Data Integration, *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 25, n. 3, pp. 39–45.
10. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M. (2002). Data Integration under Integrity Constraints. *Proceedings of CAiSE 2002*, pp. 262–279, Vol. 2348, Springer LNCS.
11. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M. (2002). On the Expressive Power of Data Integration Systems. *Proc. of the 21st Int. Conf. on Conceptual Modeling (ER 2002)*, pp. 338–350, Vol. 2503, Springer LNCS.
12. Chakravarthy, U. S., Grant, J., Minker, J. (1987). Foundations of semantic query optimization for deductive databases. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed., Morgan Kaufmann.
13. Christiansen, H., Martinenghi, D., Simplification of database integrity constraints revisited: A transformational approach. Presented at LOPSTR, Uppsala, Sweden, 25–27 august 2003. Preliminary version available at `http://www.dat.ruc.dk/~henning/LOPSTR03.pdf`
14. Dijkstra, E.W. (1976). *A Discipline of Programming*. Prentice-Hall.
15. Flach. P., Kakas, A., (eds.), *Abductive and Inductive Reasoning: Essays on their Relation and Integration*, Kluwer Academic Publishers. pp. 195–211, 2000.

oarieli@mta.ac.il

16. Godfrey, P., Grant, J., Gryz, J., Minker, J. (1988). Integrity Constraints: Semantics and Applications. *Logics for Databases and Information System*, Eds. Chomicki, J. Saake, G., Kluwer, pp. 265–306.
17. Grant, J., Minker, J. (2002). A logic-based approach to data integration. *Theory and Practice of Logic Programming (TPLP)*, vol. 2, pp. 323–368.
18. Hoare, C.A.R. (1969). An axiomatic basis for computer programming. *Communications of the ACM* 12, no. 10. pp. 576–580.
19. Kakas, A.A., Kowalski, R.A., Toni, F. (1998). The role of abduction in logic programming, *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 5, Gabbay, D.M, Hogger, C.J., Robinson, J.A., (eds.), Oxford University Press, pp. 235–324.
20. Levy, A. Y. (1999). Combining Artificial Intelligence and Databases for Data Integration. *Artificial Intelligence Today*, LNAI 1600. Eds. Wooldridge, M. J., Veloso, M., Springer-Verlag Berlin Heidelberg, pp. 249–268.
21. Li, C. (2003). Describing and Utilizing Constraints to Answer Queries in Data-Integration Systems. *IJCAI 2003 workshop on Information Integration on the Web*, On-line proceedings available at `http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm`.
22. Lin, J., Mendelzon, A. (1998). Merging Databases Under Constraints. *International Journal of Cooperative Information Systems* 7, no. 1. pp. 55–76.
23. Martinenghi, D. (2003). A Simplification Procedure for Integrity Constraints. *World Wide Web*, `http://www.dat.ruc.dk/~dm/spic/index.html`.
24. Nicolas, J.-M. (1982). Logic for Improving Integrity Checking in Relational Data Bases., *Acta Informatica* 18, pp. 227–253.
25. Nilsson, U., Małuzyński, J. (1995). *Logic, Programming and Prolog (2nd ed.)*, John Wiley & Sons Ltd.
26. Revesz, P. (1997). On the Semantics of Arbitration. *Journal of Algebra and Computation*, 7(2), pp. 133–160.
27. Rahm, E., Bernstein, P. (2001). A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4): 334–350.
28. Ullman, J. (1997). Information integration using logical views. *International Conference on Database Theory*, pp. 19–40.
29. Xu, L., Embley, D. (2003). Discovering Direct and Indirect Matches for Schema Elements. *Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), Kyoto, Japan*, pp. 39–46, IEEE Computer Society.

oarieli@mta.ac.il

On the Security of Individual Data*

János Demetrovics[1], Gyula O.H. Katona[2], and Dezső Miklós[2]

[1] Computer and Automation Institute,
Hungarian Academy of Sciences
Kende u. 13-17, H-1111, Hungary
dj@ilab.sztaki.hu
[2] Alfréd Rényi Institute of Mathematics,
Hungarian Academy of Sciences
Budapest P.O.B. 127 H-1364 Hungary
{ohkatona,dezso}@renyi.hu

Abstract. We will consider the following problem in this paper:
Assume there are n numeric data $\{x_1, x_2, \ldots, x_n\}$ (like salaries of n individuals) stored in a database and some subsums of these numbers are disclosed by making public or just available for persons not eligible to learn the original data. Our motivating question is: at most how many of these subsums may be disclosed such that none of the numbers $x_1, x_2, \ldots, x_n$ can be uniquely determined from these sums. These types of problems arise in the cases when certain tasks concerning a database are done by subcontractors who are not eligible to learn the elements of the database, but naturally should be given some data to fulfill there task. In database theory such examples are called *statistical databases* as they are used for statistical purposes and no individual data are supposed to be obtained using a restricted list of SUM queries. This problem was originally introduced by Chin and Ozsoyoglu [1], originally solved by Miller *et al.* [5] and revisited by Griggs [4].
It turned out [5] that the problem is equivalent to the following question: If there are n real, non-zero numbers $X = \{x_1, x_2, \ldots, x_n\}$ given, what is the maximum number of 0 subsums of it, that is, what is the maximum number of the subsets of X whose elements sum up to 0. This approach, together with the Sperner theorem shows that no more than $\binom{n}{n/2}$ subsums of a given set of secure data may be disclosed without disclosing at least one of the data, which upper bound is sharp as well.
However, it is natural to assume that the disclosed subsums of the original elements of the database will contain only a limited number of elements, say at most k (in the applications databases are usually huge, while the number of operations is in most of the cases limited). We have now the same question: at most how many of these subsums of at most k members may be given such that none of the numbers $x_1, x_2, \ldots, x_n$ can be uniquely determined from these sums. The main result of this paper gives an upper bound on this number, which turns out to be sharp if we

* The work was supported by the Hungarian National Foundation for Scientific Research grant numbers 37846 and 42706 and the European Community's Centre of Excellence Grant numbers ICA1-CT-2000-70009 and ICA1-CT-2000-70025.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 49–58, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

allow subsums of only k or $k-1$ members and asymptotically sharp in case of subsums of at most k members.

1 Introduction

The security of statistical databases has been studied for a long time. In this case the database is only used to obtain statistical information and therefore no individual data is supposed to be obtained as a result of the performed queries. Of course, the user is not allowed to query individual records, still, using only statistical types of queries, it might be possible to make inferences about the individual records. Several authors investigated earlier the possibility of introducing restriction for the prevention of database compromise, which include data and response perturbation, data swapping, random response queries, etc. One of the natural restrictions is to allow only SUM queries, that is queries which return the sum of the attributes corresponding to a set of individuals characterized by *characteristic formula*. For more detailed explanation of these terms see Denning [2,3]. In all of these cases it was assumed and will be assumed throughout of this paper as well that outside user or attacker do not have any further information about the database, only the answers to the SUM queries (e.g. they don't know about any functional dependencies).

Chin and Ozsoyoglu [1] introduced an Audit Expert mechanism for the prevention of database compromise with SUM queries. Later Miller *et al.* [5] determined the maximum number of SUM queries for this mechanism, which is $\binom{n}{\lfloor n/2 \rfloor}$. For example, in the database below one can ask the sum of the salaries of the individuals chosen the same number ($i = 0, 1, 2, 3$) of them from both of the sets {Bush, Carter, Clinton} and {Johnson, Kennedy, Nixon, Reagan}. In such a way one will chose $\binom{3}{0} \times \binom{4}{0} + \binom{3}{1} \times \binom{4}{1} + \binom{3}{2} \times \binom{4}{2} + \binom{3}{3} \times \binom{4}{3} = 1 + 3 \times 4 + 3 \times 6 + 1 \times 4 = 35 = \binom{7}{3}$ queries. Clearly, the given database and the one obtained from this one by lowering the salaries of {Bush, Carter, Clinton} by 1000 and increasing the salaries of {Johnson, Kennedy, Nixon, Reagan} by 1000 will give exactly the same answer to these queries and therefore no individual salary can be exactly calculated from this set of questions.

Table 1. Sample Database

Name	*Salary*
Bush	250000
Carter	180000
Clinton	220000
Johnson	120000
Kennedy	100000
Nixon	140000
Reagan	160000

oarieli@mta.ac.il

A natural restriction of the above question is the restriction of the size of the SUM queries, that is assuming that the sums may involve at most or exactly k members. E.g., if in the above database we only consider SUM queries summing up 3 data, a possible scheme of them without compromising the database is to ask the some of the salaries of 3 gentlemen, two chosen from the set {Bush, Carter, Clinton, Johnson, Kennedy} and one from the set {Nixon, Reagan}. Therefore altogether $\binom{5}{2} \times \binom{2}{1} = 20$ queries are made, and, again, by increasing the salaries of {Bush, Carter, Clinton, Johnson, Kennedy} and decreasing the salaries of {Nixon, Reagan} with the double of that amount shows that no individual data can be gained from this set of statistical queries.

The main results of the recent paper, presented is Section 3, Theorems 3.1 and 3.2 answer the questions about the maximal possible SUM queries when either only a given number of data can be summed any time or when the number of the data involved in any SUM queries is bounded above. The first question is solved completely — that is a construction of the possible sequence of queries, the number of them equal to the obtained upper bound, is given — assuming (what can be quite natural in the real use of databases) that the number of records is much larger then the allowed number of them in the SUM queries. The second case is answered asymptotically.

In Section 2 we will carry on a sequence of transformations of the original questions, most of the repeated (or simply referred to) the transformations done by Chin and Ozsoyoglu [1] and Miller *et al.* [5,6] to formulate the exact mathematical questions to be solved in Section 3. In Section 4, we will draw the conclusions to answer the original statistical database questions.

2 Deriving the Mathematical Problems

Let us be given n real numbers $\{x_1, x_2, \ldots, x_n\}$ (like salaries of n individuals in the sample database) stored in a database. A possible SUM query is to ask $\sum_{i \in A} x_i$ for some $A \subset X = \{1, 2, 3, \ldots, n\}$ and we would like to maximize the number of these queries (maybe with some other side constraints) such that they will not determine any of the original data x_i's. That is we would like to give a sequence of subsets of X, $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$, maximize m, such that the sums $\left\{\sum_{i \in A_j} x_i : 1 \leq j \leq m\right\}$ do not determine any of the x_i's. We will only consider restricted type of attacks, that is methods to calculate the values of x_i's from the known sums, namely linear combinations. However, the upper bound proven for this restricted type of attacks will turn out to be sharp for the general case as well. In Section 4 we will give constructions of databases together with the sequence of SUM queries such that their number will be equal to the obtained maximum (if we only assume linear combination attacks) and the different databases (all individual data will be pairwise different) will both give the same answer to these SUM queries.

To formulate the problem in another, for our investigation more suitable way, consider the n dimensional vector space over the real numbers, $\mathbf{R}^n$, and the unit

oarieli@mta.ac.il

vectors $\mathbf{e}_i = \{\overset{1}{0}, \overset{2}{0}, \ldots, \overset{i}{1}, \ldots, \overset{n}{0}\}$, $i = 1, 2, \ldots, n$. Denote the characteristic vectors of the subsets A_i by $\mathbf{v}_i$, that is $\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots v_{in})$, where $v_{ij} = 1$ iff $x_j \in A_i$, 0 otherwise. With this setting, we are looking for the maximum number of $\mathbf{v}_i$'s such that none of the unit vectors $\mathbf{e}_i$'s are in $\langle \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \rangle$, the subspace spanned by the m characteristic vectors of the subsets determining the members of the subsums. This can easily be seen with the following straightforward lemma.

Lemma 2.1 *Let* $\mathbf{x}$ *denote the vector* $\{x_1, x_2, \ldots, x_n\}$ *and for given sequence of SUM queries with characteristic vectors* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ *consider the vectors* $\mathbf{v}$ *where the value* $\mathbf{vx}$*, the scalar product of the two vectors* $\mathbf{v}$ *and* $\mathbf{x}$*, can be calculated from the values* $\mathbf{v}_i\mathbf{x}$*, that is,* $\mathbf{vx}$ *is uniquely determined by the two vectors* $\mathbf{v}$ *and* $\mathbf{x}$*. Then these vectors will form a subspace of the original vector space equal to* $\langle \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \rangle$*.* □

From now on, instead of the sequence of the SUM queries we will consider the subspace spanned by the characteristic vectors. Any question regarding the maximum number of queries with certain property is equivalent to the question of the maximum number of the $(0, 1)$-vectors (with the additional required properties) of the subspace not containing any of the unit vectors.

The following further reduction steps of the problem are originally due to Chin and Ozsoyoglu [1].

Lemma 2.2 *(***Chin and Ozsoyoglu [1]***) If* $\mathbf{V} \subset \mathbf{R}^n$*,* $\mathbf{e}_i \notin \mathbf{V}$ $1 \leq i \leq n$*,* $\dim \mathbf{V} \leq (n-1)$*, then there is a* $\mathbf{W} \supset \mathbf{V}$ *such that* $\dim \mathbf{W} = n-1$*,* $\mathbf{e}_i \notin \mathbf{W}$ $1 \leq i \leq n$*.* □

Since any n (full) dimensional space would contain all unit vectors $\mathbf{e}_i$, and — by the lemma above — all at most $n-1$ dimensional spaces will be contained by an exactly $n-1$ dimensional space having the required property, we may assume that the subspace giving the maximum possible number of allowed queries is $n-1$ dimensional. Take the matrix of a basis of this subspace and bring it to it normal form

$$\begin{vmatrix} 1\,0 \cdots 0 & a_1 \\ 0\,1 \cdots 0 & a_2 \\ \vdots\,\vdots\,\ddots\,\vdots & \vdots \\ 0\,0 \cdots 1 & a_{n-1} \end{vmatrix}$$

where none of the a_i's are equal to 0 due to the fact that the unit vectors are not in the subspace.

The subspace spanned by the characteristic vectors of the allowed SUM queries is also spanned by the rows of this matrix. Therefore all of these characteristic vectors are in the subspace spanned by the rows of the matrix. On the other hand, all the $0, 1$ vectors being in the subspace spanned by the rows of the matrix do determine a SUM query and (if they satisfy the further assumptions, like the number of 1's is k or at most k) they are allowed, that is the set of them will not compromise the database.

oarieli@mta.ac.il

It is easy to see that the only linear combinations of the rows of this matrix yielding $(0,1)$-vectors are those with coefficients $0, 1$. Any such linear combination will be a $0,1$ vector if and only if the sum $\sum a_i$ over all i where the corresponding coefficient is 1 is either 0 or 1. Therefore, we have to maximize the number of sums $\sum_{i\in A} a_i = 0$ or 1 where the A's are subsets of $[n-1] = \{1, 2, \ldots, n-1\}$. Let us introduce $a_n = -1$ and now consider the sums $\sum_{i\in B} a_i = 0$ where the B's are subsets of $[n] = \{1, 2, \ldots, n\}$. Naturally, there is a one-to-one correspondence between these two sets of sums. Therefore, our original question

Problem 2.3 *Determine the maximum possible number of SUM queries over a set of n records without compromising the database.*

is now reduced to the following one:

Problem 2.4 *Given a set of n real numbers $\{a_1, a_2, \ldots, a_n\}$, none of them being equal to 0, determine the maximum number of sums $\sum_{i\in B} a_i = 0$ where the B's are subsets of $[n] = \{1, 2, \ldots, n\}$.*

Further, if we assume that the number of elements in the SUM queries are restricted by a size constraint (like at most or exactly k element subsets are only considered), the same restriction will apply to the sums in Problem 2.4.

In Problem 2.4 we omitted the assumption that $a_n = -1$ since any set of n non-zero real numbers $\{y_1, y_2, \ldots, y_n\}$ can be normalized (simply each of them multiplied by the same non-zero number) such that for the resulting vector $\{x_1, x_2, \ldots, x_n\}$ we will have $x_n = -1$ and the set (and therefore the number) of zero sums naturally will not change.

Now consider the set of real numbers $\{a_1, a_2, \ldots, a_n\}$ and the system of subsets of the indices $X = \{1, 2, \ldots, n\}$, $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$ such that the sums $\sum_{i\in B_j} a_i$ are all 0. Let X_1 be the set of indices of the positive a_i's and X_2 be the set of the indices of the negative a_i's. Since none of the a_i's are zero, $X = X_1 \cup X_2$ is a partition of the set X. If we consider two sets B_1 and B_2 from the set $\mathcal{B}$, then since $\sum_{i\in B_1} a_i = \sum_{i\in B_2} a_i = 0$, we have $\sum_{i\in B_1 - B_2} a_i = -\sum_{i\in B_1\cap B_2} a_i = \sum_{i\in B_2 - B_1} a_i$ and therefore the sums at the two ends of this system of equations are equal and so have the same sign. Therefore, it is not possible that $B_1 - B_2 \subset X_1$ and $B_2 - B_1 \subset X_2$ at the same time.

Definition 2.5 *Let $X = X_1 \cup X_2$ be a partition of the finite set X and F and G two subsets of X. We say that F and G are separated by the partition if $F - G \subset X_1$ and $G - F \subset X_2$ does not happen at the same time.*

We also know that all of the sets B_i have the property $B_i \cap X_1 \neq \emptyset$ and $B_i \cap X_2 \neq \emptyset$, since sum of only negative or only positive numbers may not be equal to zero.

Definition 2.6 *We say that the family $\mathcal{F}$ of subsets of the finite set $X = X_1 \cup X_2$ ($X_1 \cap X_2 = \emptyset$, $X_1 \neq \emptyset$, $X_2 \neq \emptyset$) is difference separated (with respect to the partition $X_1 \cup X_2$) if $F - G$ and $G - F$ are separated by the partition for every pair F, G of distinct members of $\mathcal{F}$ and $F \cap X_1 \neq \emptyset, F \cap X_2 \neq \emptyset$ holds for each member.*

oarieli@mta.ac.il

We know by now that for any set of SUM queries not compromising the database we can find a set of a set of n real non-zero numbers $\{a_1, a_2, \ldots, a_n\}$, such that the subsets of indices indices $X = \{1, 2, \ldots, n\}$, $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$ for which the sums $\sum_{i \in B_j} a_i$ are 0 do correspond to the SUM queries, on one side, and form a difference separated family of subsets on the other side. That is, any upper bound on the size of difference separated families (in our case of subsets of given sizes) will give an upper bound on the possible number of SUM queries not compromising the database. In the next section we will derive such upper bounds, while in the last section — Conclusions — with giving specific examples we will also show that this bounds are sharp not only for the size of the difference separated families but for the number of SUM queries as well.

3 The Main Mathematical Theorems

Theorem 3.1 *Let $0 < k, n$ be fixed even integers, $n_0(k) \leq n$, that is n is large relative to k. Let further X be an n-element set with partition $X = X_1 \cup X_2$ where $X_1, X_2 \neq \emptyset$. Suppose that $\mathcal{F}$ is a difference separated family (with respect to $X_1 \cup X_2$) of subsets of size k and $k-1$. Then*

$$|\mathcal{F}| \leq M(n,k) = \binom{\left\lfloor \frac{(n+1)(k-1)}{k} \right\rfloor}{k-1} \left(n - \left\lfloor \frac{(n+1)(k-1)}{k} \right\rfloor \right). \tag{1}$$

Theorem 3.2 *Let $0 < k, n$ be fixed even integers, $n_0(k) \leq n$, that is n is large relative to k. Let further X be an n-element set with partition $X = X_1 \cup X_2$ where $X_1, X_2 \neq \emptyset$. Suppose that $\mathcal{F}$ is a difference separated family (with respect to $X_1 \cup X_2$) of subsets of size at most k. Then*

$$|\mathcal{F}| \leq M(n,k) + M(n,k-2) + M(n,k-4) + \ldots.$$

Remark. Theorem 3.1 is sharp, since choosing X_1 to have $\left\lfloor \frac{(n+1)(k-1)}{k} \right\rfloor$ elements and taking all $k-1$-element subsets of X_1 combining them with all possible 1-element sets in X_2, the number of sets will be exactly $M(n,k)$. On the other hand this construction obviously satisfies the conditions. Theorem 3.2, however is not necessarily sharp, since the obvious generalization of the construction above does not always work. It is, however, asymptotically sharp, since $M(n,k-2) + M(n,k-4) + \ldots = O(n^{k-3})$ while $M(n,k)$ is of order $k-1$.

The proof will be given by a sequence of lemmas.

Assume that a cyclic ordering is fixed both in X_1 and X_2. We say that the pair (A,B) of subsets $A \subset X_1, B \subset X_2$ is an (a,b)-*interval-pair* if A and B are intervals in the respective X and $|A| = a, |B| = b$. The *middle pair* of the (a,b)-interval-pair (A,B) is (x,y) where x is the $\lfloor \frac{a+1}{2} \rfloor$th element of A and y is the $\lfloor \frac{b+1}{2} \rfloor$th element of B.

Lemma 3.3 *Suppose that (A,B) and (C,D) are (a,b) and (c,d)-interval-pairs, respectively, where $a+b = c+d$ (a,b,c,d are positive integers). If their middle*

oarieli@mta.ac.il

pairs coincide then either $A \cup B - C \cup D \subset X_1$ *and* $C \cup B - A \cup B \subset X_2$ *or* $A \cup B - C \cup D \subset X_2$ *and* $C \cup B - A \cup B \subset X_1$ *hold.*

Proof. It is easy to see that if $c \leq a$ then $C \subset A$. The inequality $d \geq b$ is a consequence, this implies $B \subset D$. Hence we have $A \cup B - C \cup D \subset X_1$ and $C \cup B - A \cup B \subset X_2$. The case $c > a$ is analogous. □

Lemma 3.4 *Suppose that* (A, B) *and* (C, D) *are* (a, b) *and* (c, d)*-interval-pairs, respectively, where* $a+b = c+d\pm 1$ *(*a, b, c, d *are positive integers). If their middle pairs coincide then either* $A \cup B - C \cup D \subset X_1$ *and* $C \cup B - A \cup B \subset X_2$ *or* $A \cup B - C \cup D \subset X_2$ *and* $C \cup B - A \cup B \subset X_1$ *hold.*

Proof. The previous proof can be repeated. □

Lemma 3.5 *Let* $\mathcal{G}$ *be a family of difference separated intervals with respect to* $X_1 \cup X_2$ *with members of size* $j - 1$ *and* j $(2 \leq j)$*. Then* $|\mathcal{G}| \leq n_1 n_2$ *holds.*

Proof. The members of $\mathcal{G}$ must have different middle pairs by Lemmas 3.2 and 3.3. The number of possible middle pairs is $n_1 n_2$. □

Introduce the following definition:

$$M(n_1, n_2; k) = \max\left(\binom{n_1}{i}\binom{n_2}{j}\right) \tag{2}$$

where the maximum is taken for all $0 < i \leq n_1, 0 < j \leq n_2, i + j = k$.

Lemma 3.6 *Let* $X = X_1 \cup X_2$, $X_1 \cap X_2 = \emptyset$, $|X_1| = n_1$, $|X_2| = n_2$*. If* $\mathcal{F}$ *is a family of difference separated sets of sizes* ℓ *and* $\ell - 1$ *then*

$$|\mathcal{F}| \leq M(n_1, n_2; \ell) \tag{3}$$

holds.

Proof. The number of four-tuples $(\mathcal{C}_1, \mathcal{C}_2, A, B)$ will be counted in two different ways, where $\mathcal{C}_i$ is a cyclic permutation of X_i $(i = 1, 2)$, $A = F \cap X_1$ and $B = F \cap X_2$ holds for some $F \in \mathcal{F}$ and they form an interval-pair for these cyclic permutations.

Let first fix A and B and count the number of cyclic permutations where they are intervals. $\mathcal{C}_1$ can be chosen in $|A|!(n_1 - |A|)!$ many ways, the same applies for B, therefore the number of four-tuples is

$$\sum |A|!(n_1 - |A|)!|B|!(n_2 - |B|)! \tag{4}$$

where the summation is taken for all $A = F \cap X_1, B = F \cap X_2, F \in \mathcal{F}$.

Fix now the cyclic permutations. The subfamily of $\mathcal{F}$ consisting of the interval-pairs for these permutations will be denoted by $\mathcal{G}$. It is a family of difference separated intervals. The application of Lemma 3.4 gives that the numbers of pairs A, B for any given pair of cyclic permutations is at most $n_1 n_2$. Since

oarieli@mta.ac.il

the number of permutations is $(n_1-1)!(n_2-1)!$, the number of four-tuples in question is at most $n_1!n_2!$. Compare it with (4):

$$\sum |A|!(n_1-|A|)!|B|!(n_2-|B|)! \le n_1!n_2!.$$

Elementary operations lead to

$$\sum \frac{1}{\binom{n_1}{|A|}\binom{n_2}{|B|}} \le 1 \tag{5}$$

where $A = F \cap X_1, B = F \cap X_2, 0 < |A| \le n_1, 0 < |B| \le n_2, |A|+|B| = \ell$ or $\ell-1$. Since $M(n_1,n_2;\ell-1) \le M(n_1,n_2;\ell)$ holds, by the definition of $M(n_1,n_2;\ell)$ (5) implies

$$\frac{|\mathcal{F}|}{M(n_1,n_2;\ell)} \le 1$$

proving (3). □

Lemma 3.7 *Suppose* $1 \le i < \ell \le n, \ell - i < n - n_1$. *Then*

$$\binom{n_1}{i}\binom{n-n_1}{\ell-i} \le \binom{\left\lfloor \frac{(n+1)i}{\ell} \right\rfloor}{i}\binom{n-\left\lfloor \frac{(n+1)i}{\ell} \right\rfloor}{\ell-i}.$$

Proof. Compare two consecutive expressions:

$$\binom{n_1}{i}\binom{n-n_1}{\ell-i} \le \binom{n_1+1}{i}\binom{n-n_1-1}{\ell-i}.$$

After carrying out the possible cancellations

$$\frac{n-n_1}{n-n_1-\ell+i} \le \frac{n_1+1}{n_1-i+1}$$

is obtained what is equivalent to

$$n_1+1 \le \frac{(n+1)i}{\ell}.$$

Hence

$$\binom{n_1}{i}\binom{n-n_1}{\ell-i}$$

takes on its maximum (with fixed i and ℓ) at

$$\left\lfloor \frac{(n+1)i}{\ell} \right\rfloor.$$

□

Lemma 3.8 *Suppose* $0 < \frac{\ell}{2} \le i \le \ell - 1$. *If* $n_0(\ell) \le n$ *then*

$$\binom{\left\lfloor \frac{(n+1)i}{\ell} \right\rfloor}{i}\binom{n-\left\lfloor \frac{(n+1)i}{\ell} \right\rfloor}{\ell-i} < \binom{\left\lfloor \frac{(n+1)(i+1)}{\ell} \right\rfloor}{(i+1)}\binom{n-\left\lfloor \frac{(n+1)(i+1)}{\ell} \right\rfloor}{\ell-i-1}.$$

oarieli@mta.ac.il

Proof. After carrying out the possible cancellations,

$$\frac{\left(n-\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor\right)\dots\left(n-\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor+1\right)}{(\ell-i)\left(\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor-i\right)}<$$

$$\frac{\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor\dots\left(\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor+1\right)}{(i+1)\left(n-\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor-\ell+i+1\right)}$$

is obtained what is equivalent to

$$\frac{\left(n-\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor\right)\dots\left(n-\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor+1\right)}{\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor\dots\left(\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor+1\right)}<$$

$$\frac{(\ell-i)\left(\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor-i\right)}{(i+1)\left(n-\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor-\ell+i+1\right)} \tag{6}$$

Consider the left hand side as a product of quotients: the quotient of the first factors, the second factors, etc., in the numerator and the denominator, respectively. The first of these quotients is the largest one, it is less than 1, their number is at least $\frac{n+1}{\ell}$. Therefore the left hand side in (6) can be replaced by

$$\left(\frac{n-\left\lfloor\frac{(n+1)i}{\ell}\right\rfloor}{\left\lfloor\frac{(n+1)(i+1)}{\ell}\right\rfloor}\right)^{\frac{n+1}{\ell}}.$$

A further increase is

$$\left(\frac{n+1-\frac{(n+1)i}{\ell}}{\frac{(n+1)(i+1)}{\ell}}\right)^{\frac{n+1}{\ell}}=\left(\frac{\ell-i}{i+1}\right)^{\frac{n+1}{\ell}}.$$

Here $\ell-i<i+1$ by the assumption of the lemma, therefore the left hand side of (6) tends exponentially to 0 when n tends to infinity. On the other hand, the right hand side of (6) tends to

$$\frac{(\ell-i)i}{(i+1)(\ell-i-1)},$$

proving the inequality. □

Remark. This lemma seems to be true for small values of n, too, we have technical difficulties to prove it.

Proof of Theorem 3.1. By Lemma 3.6 $|\mathcal{F}|$ cannot exceed the largest product $\binom{n_1}{i}\binom{n-n_1}{k-i}$ where $1\le n_1<n, 1\le i\le n_1, 1\le k-i\le n-n_1$. By symmetry $\frac{\ell}{2}\le i$ can be supposed. Lemma 3.7 gives the best n_1. This upper estimate in Lemma 3.7 can be increased by Lemma 3.8 until we arrive to the largest possible value of i: $\ell-1$. This is the desired upper estimate. □

Proof of Theorem 3.2. Apply Theorem 3.1 with $k, k-2, k-4, \dots$ and sum the obtained upper estimates. □

oarieli@mta.ac.il

4 Conclusion

By the argument of Section 2 and the results of Section 3, if $n_0(k) < n$, at most $M(n,k)$ (see (1) for exact value) SUM queries of size k can be asked about a set of data $x_1, x_2, \dots x_n$ without disclosing at least one of the values x_i. On the other hand, this bound is not only sharp for the mathematical questions asked in the previous section, but also for the original problem. Assume n equal real numbers divided into two parts: B_1 of size $\left\lfloor \frac{(n+1)(k-1)}{k} \right\rfloor$ and B_2 of size $\left(n - \left\lfloor \frac{(n+1)(k-1)}{k} \right\rfloor\right)$. Take all subsums of this numbers of k elements such that $k-1$ are chosen from set B_1 and 1 from set B_2. Now increase all of the elements of B_1 by 1 and decrease all of the elements of B_2 by $k-1$ (assume that originally the common value was not -1 neither $k-1$) and take exactly the same subsums. The answers to these queries are the same in both of cases, that is these answers do not disclose any of the values x_i's.

It is also interesting to mention that the bound $M(n,k)$ is only constant time smaller than the absolute upper bound $\binom{n}{k}$ for the number of SUM queries of size k, and much bigger than the somewhat more general solution of $\binom{n/2}{k/2}^2$, which is $\sqrt{k}$ time smaller than $\binom{n}{k}$. One can get a construction yielding the $\binom{n/2}{k/2}^2$ bound simply using the above method but dividing the set of the values into two equal size subsets and picking always the same number of elements from both sides. For example, if $n = 20$ and $k = 10$, then $\binom{n}{k} = 184756$, $M(n,k) = 97240$ and $\binom{n/2}{k/2}^2 = 63504$.

The bound for the case when the SUM queries may have any number of elements less than or equal to k is most probably not sharp. At the same time, similar to the case in the mathematical theorem, an asymptotically good construction can be given simply taking the above construction for the case when all sums have exactly k elements (see Remark after Theorem 3.2.)

References

1. Chin, F.Y., Ozsoyoglu, G., Auditing and inference control in statistical databases, *IEEE Transactions on Software Engineering* **SE-8**(1982) 574–582.
2. Denning, D.E., in "Cryptography and Data Security", Addison-Wesley, Sydney, 1982.
3. Denning, D.E., Schlorer, J., Inference controls for statistical databases, *Computer* (1983), 69–82.
4. Griggs, J.R., Concentrating subset sums at k points, *Bull. Inst. Comb. Applns.* **20**(1997) 65–74.
5. Miller, M., Roberts, I., Simpson I., Application of Symnmetric Chains to an Optimization Problem in the Security of Statistical Databases, *Bull. ICA* **2**(1991) 47–58.
6. Miller, M., Roberts, I., Simpson I., Preventation of Relative Compromise in Statistical Databases Using Audit Expert, *Bull. ICA* **10**(1994) 51–62.

oarieli@mta.ac.il

Implementing Ordered Choice Logic Programming Using Answer Set Solvers

Marina De Vos*

Department of Computer Science
University of Bath
Bath, United Kingdom
mdv@cs.bath.ac.uk

Abstract. Ordered Choice Logic Programming (OCLP) allows for dynamic preference-based decision-making with multiple alternatives without the need for any form of negation. This complete absence of negation does not weaken the language as both forms (classical and as-failure) can be intuitively simulated in the language and eliminated using a simple pre-processor, making it also an easy language for users less familiar with logic programming. The semantics of the language is based on the preference between alternatives, yielding both a skeptical and a credulous approach. In this paper we demonstrate how OCLPs can be translated to semi-negative logic programs such that, depending on the transformation, the answer sets of the latter correspond with the skeptical/credulous answer sets of the former. By providing such a mapping, we have a mechanism for implementing OCLP using answer set solvers like Smodels or dlv. We end with a discussion of the complexity of our system and the reasoning tasks it can perform.

1 Introduction

Examining human reasoning, we find that people often use preference, order or defaults for making decisions: "I prefer this dish", "This colour goes better with the interior", "This item costs more", "In general, the human heart is positioned at the left". When faced with conflicting information, one tends to make decisions that prefer an alternative corresponding to more reliable, more complete, more preferred or more specific information. When modelling knowledge or non-monotonic reasoning via computer programs, it is only natural to incorporate such mechanisms.

In recent years several proposals for the explicit representation of preference in logic programming formalisms have been put forward. [19,18] are just two examples.

Systems that support preferences find applications in various domains such as law, object orientation, scheduling, model based diagnosis and configuration tasks. However, most approaches use preferences only when the models have already been computed, i.e. decisions have already been made, or only support preferences between rules with opposite (contradictory) consequences, thus statically limiting the number of alternatives of a decision.

* This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging technologies under the IST-2001-37004 WASP project.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 59–77, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

We propose a formalism ([16]), called ordered choice logic programming, that enables one to dynamically reason about situation-dependent decisions involving multiple alternatives. The dynamics of this system is demonstrated by the following example.

Example 1.1. Buying a laptop computer involves a compromise between what is desirable and what is affordable. Take, for example, the choice between a CD, CDRW or DVD drive. The CD is the cheaper option. On the other hand, for a laptop, a DVD drive may be more useful than a CD writer. If the budget is large enough, one could even buy two of the devices. The above information leads one to consider two possible situations.

- With a smaller budget, a DVD-player is indicated, while
- with a larger budget, one can order both a DVD-player and a CD-writer.

To allow this kind of reasoning, a program consists of a (strict) partially ordered set of components containing choice rules (rules with exclusive disjunction in the head). Information flows from less specific components to the more preferred ones until a conflict among alternatives arises, in which case the most specific one will be favoured. The situation becomes less clear when two alternatives are equally valued or are unrelated. The decision in this case is very situation dependent: a doctor having a choice between two equally effective cures has to make a decision, while you better remain indecisive when two of your friends have an argument! To allow both types of intuitive reasoning, two semantics are introduced: a credulous and a more skeptical one.

OCLP provides an elegant and intuitive way of representing and dealing with decisions. People with little or no experience with non-monotonic reasoning can easily relate to it, due to the absence of negation. This absence of negation does not restrict the language in any way, as both types (classic and as-failure) can easily be simulated. When users insist they can use negation, a simple pre-processor can then be used to remove it while maintaining the semantics.

In computer science, having a nice theory alone is not enough; one also needs to be able to apply it. The aim of this paper is to provide the theoretical foundations for an implementation.

In this paper, we investigate the possibility for building an OCLP front-end for answer set solvers. Smodels ([20]), developed at Helsinki University of Technology, and dlv ([26]), created at the Technical University of Vienna and the University of Calabria are currently the most popular ones. An initial implementation build on top of Smodels can be obtained from `http://www.cs.bath.ac.uk/$\sim$mdv/oct/`.

The remainder of this paper is organised as follows: in Section2 we continue with a short overview of the basic notions concerning choice logic programming, the language behind OCLP. Section 3 focuses on the introduction of OCLP with its skeptical and credulous answer set semantics. Section 4 deals with two mapping of OCLPs to semi-negative logic programs allowing answer set solvers to work with OCLP. We also take a closer look at the complexity of the proposed mappings. These transformations, one for each semantics, can then serve as a theoretical model for our front-end. We investigate various ways how we could, implementation wise, improve them. Furthermore, we introduce an initial implementation, called OCT, which computes both types of answer sets on top of Smodels. When explaining the theoretical aspects of both OCLP and the mappings we always considered our programs to be grounded. At the end of the section, we have a closer look at the technical issues that arise when we extend to

oarieli@mta.ac.il

the non-grounded case with a finite Herbrand Universe. In Section 6, we investigate the complexity and the expressive power of our language. We end this paper with a discussion on the relations with other approaches (Section 7) and directions for future research (Section 8).

2 Choice Logic Programming

Choice logic programs [15] represent decisions by interpreting the head of a rule as an exclusive choice between alternatives.

Formally, a *Choice Logic Program* [15], CLP for short, is a countable set of rules of the form $A \leftarrow B$ where A and B are finite sets of ground atoms. Intuitively, atoms in A are assumed to be xor'ed together while B is read as a conjunction (note that A may be empty, i.e. constraints are allowed). The set A is called the head of the rule r, denoted H_r, while B is its body, denoted B_r. In examples, we use "$\oplus$" to denote exclusive disjunction[1], while "," is used to denote conjunction.

The *Herbrand base* of a CLP P, denoted $\mathcal{B}_P$, is the set of all atoms that appear in P. An *interpretation*[2] is a subset of $\mathcal{B}_P$.

A rule r in a CLP is said to be *applicable* w.r.t. an interpretation I if $B_r \subseteq I$. Since we are modelling choice, we have that r is *applied* when r is applicable and[3] $|H_r \cap I| = 1$. A rule is *satisfied* if it is applied or not applicable. A *model* is defined in the usual way as a total interpretation that satisfies every rule. A model M is said to be *minimal* if there does not exist a model N such that $N \subset M$.

3 Ordered Choice Logic Programming

An ordered choice logic program (OCLP) is a collection of choice logic programs, called components, which are organised in a strict partial order[4] that represents some preference criterion (e.g. specificity, reliability, ...).

Definition 3.1. *An **Ordered Choice Logic Program**, or OCLP, is a pair $\langle \mathcal{C}, \prec \rangle$ where $\mathcal{C}$ is a finite set of choice logic programs, called **components**, and "$\prec$" is a strict pointed partial order on $\mathcal{C}$.*

For two components $C_1, C_2 \in \mathcal{C}$, $C_1 \prec C_2$ implies that C_1 is preferred over C_2. Throughout the examples, we will often represent an OCLP P by means of a directed acyclic graph (dag) in which the nodes represent the components and the arcs the $\prec$-relation, where arcs point from smaller (more preferred) to larger (less preferred) components.

[1] The exclusive disjunction used here is stronger than the one used in classic propositional logic. Here we assume that $a \oplus b \oplus c$ can only be true iff exactly one atom is true. The same statement in classical logic is also true when all three atoms are true.

[2] In this paper we only work with total interpretations: each atom from the Herbrand base is either true or false. Bearing this in mind, it suffices to mention only those atoms which can be considered true.

[3] For a set X, we use $|X|$ do denote its cardinality.

[4] A relation R on a set A is a strict partial order iff R is anti-reflexive, anti-symmetric and transitive. R is pointed if an element $a \in A$ exists such that aRb for all $b \in A$.

oarieli@mta.ac.il

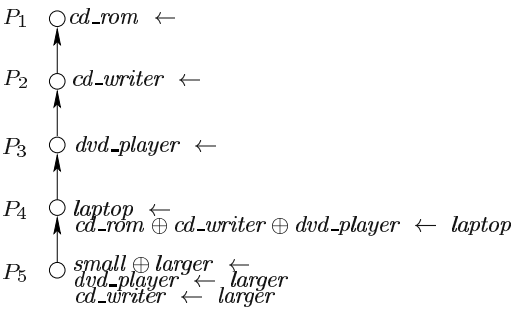


Fig. 1. The Configuration OCLP of Example 3.1

Example 3.1. The decision problem from the introduction (Example 1.1) can easily be written as an OCLP, as shown in Figure 1. The rules in components P_1, P_2 and P_3 express the preferences in case of a small budget. The rules in P_4 express the intention to buy/configure a laptop and, because of this, a decision about its various devices should be made. In component P_5, the first rule states the possibility of a larger budget. If so, the two remaining rules allow the purchase of both a DVD-player and a CD-writer.

Definition 3.2. *Let P be an OCLP. We use $P^\star$ to denote the CLP that contains all the rules appearing in (a component of) P. We assume that rules in $P^\star$ are labelled by the component from which they originate and we use $c(r)$ to denote the component*[5] *of r. The* Herbrand base *$\mathcal{B}_P$ of P is defined by $\mathcal{B}_P = \mathcal{B}_{P^\star}$.*
*An **interpretation** for P is any interpretation of $P^\star$. We say that a rule r in P is **applicable** w.r.t. an interpretation I iff $B_r \subseteq I$; r is **applied** w.r.t. I iff r is applicable and $|H_r \cap I| = 1$.*

Example 3.2. For the OCLP in Example 3.1, the sets $I = \{dvd\_player, small\}$, $J = \{laptop, cd\_writer, small\}$, $K = \{laptop, dvd\_player, small\}$ and $L = \{dvd\_player, larger, cd\_writer, cd\_player, laptop\}$ are all interpretations. The interpretation I makes the rule $small \oplus larger \leftarrow$ applied while the applicable rule $cd\_writer \leftarrow$ is not applied.

Facing a decision means making an exclusive choice between the various alternatives which are available. If we want OCLP to model/solve decision problems we need a mechanism for representing them. In a CLP, decisions are generated by so-called *choice rules* i.e. rules with multiple head atoms. For OCLP, we can do something similar as long as we also take the preference order into account. We want to make sure that we leave the option open to overrule the exclusiveness of a choice when in more preferred components multiple alternatives are suggested (e.g. Example 1.1). Hence we say that an atom a is an *alternative* for an atom b in a component C if an applicable rule exists in a component at least as preferred as C containing both a and b in its head.

[5] Without losing generality, we can assume that a rule appears in only one component.

oarieli@mta.ac.il

Definition 3.3. *Let I be an interpretation of an OCLP $P = \langle \mathcal{C}, \prec \rangle$ with $C \in \mathcal{C}$. The set of **alternatives** in C for an atom $a \in \mathcal{B}_P$ w.r.t. I, denoted $\Omega^I_C(a)$, is defined as*[6]*:* $\Omega^I_C(a) = \{b \mid \exists r \in P^\star \cdot c(r) \preccurlyeq C \ \wedge \ B_r \subseteq I \ \wedge \ a, b \in H_r \text{ with } a \neq b\}$.

Example 3.3. Reconsider Example 3.2. The alternatives for cd_rom in P_2 w.r.t. J are $\Omega^J_{P_2}(cd\_rom) = \{dvd\_player, cd\_writer\}$. W.r.t. I, we obtain $\Omega^I_{P_2}(cd\_rom) = \emptyset$, since the choice rule in P_4 is not applicable. When we take P_5 instead of P_2, we obtain w.r.t. J: $\Omega^J_{P_5}(cd\_rom) = \emptyset$.

Given the alternatives in a certain context (a component and an interpretation), one naturally selects that alternative that is motivated by a more preferred rule, thus *defeating* the rule(s) suggesting less preferred alternatives. However, if alternatives appear in the same or unrelated components, two approaches are possible: using a skeptical strategy, one would refrain from making a decision, i.e. not selecting any of the various alternatives, while a credulous setting suggests an arbitrary choice of one of the alternatives. For both types of reasoning one can think of situations where one approach works while the other gives an incorrect, unintuitive outcome. Skeptical reasoning is practiced in American law when a jury cannot come to a unanimous decision. An example of credulous reasoning is the decision a goal-keeper faces in football when trying to stop a penalty. To accommodate this problem, we introduce a semantics for both types of reasoning. From a skeptical viewpoint, we say that rule is defeated if one can find a better, more preferred alternative for each of its head atoms.

Definition 3.4. *Let I be an interpretation for an OCLP P. A rule $r \in P^\star$ is **defeated** w.r.t. I iff* $\forall a \in H_r \cdot \exists r' \in P^\star \cdot c(r') \prec c(r) \ \wedge \ B_{r'} \subseteq I \ \wedge \ H_{r'} \subseteq \Omega^I_{c(r)}(a)$.

Example 3.4. Reconsider Example 3.2. The rule $cd\_rom \leftarrow$ is defeated w.r.t. J by the rule $cd\_writer \leftarrow$. The rule $cd\_rom \oplus cd\_writer \oplus dvd\_player \leftarrow$ is defeated w.r.t. L by the combination of the rules $dvd\_player \leftarrow larger$ and $cd\_writer \leftarrow larger$.

Example 3.5. Consider the OCLP $\langle \{P_1 = \{a \leftarrow ; b \leftarrow\}, P_2 = \{a \oplus b \leftarrow\}\}, P_2 \prec P_1 \rangle$. Given the interpretation $\{b\}$, the rule $r : a \leftarrow$ is not defeated. The atom a has one alternative, i.e. b, thanks to the applicable choice rule in component P_2. However, examining the more preferred components of $c(r) = P_1$, we cannot find any rule having b in the head.

Just as for the skeptical semantics we need to define an appropriate defeating strategy for our credulous approach. An obvious way of doing so consists of simply dropping the condition that an alternative should be found in a more preferred component. Unfortunately, this leads to unintuitive results. To avoid this, we need to make sure that credulous defeaters are not only applicable, but also applied.

Definition 3.5. *Let I be an interpretation for an OCLP P. A rule $r \in P^\star$ is **c-defeated** w.r.t. I iff* $\forall a \in H_r \cdot \exists r' \in P^\star \cdot c(r) \not\prec c(r') \ \wedge \ r'$ *is applied w.r.t.* $I \ \wedge \ H_{r'} \subseteq \Omega^I_{c(r)}(a)$.

[6] $\preccurlyeq$ is the reflexive closure of $\prec$.

oarieli@mta.ac.il

Note that this definition allows the rules r' to come from a more preferred, the same or unrelated component. At first sight one might think that a situation could occur where $r = r'$. However, this is impossible as an atom can never be considered an alternative of itself.

Example 3.6. While the skeptical approach makes it impossible to have the rule $a \leftarrow$ in Example 3.5 defeated w.r.t. $\{b\}$, the credulous semantics can.

For our model semantics, both skeptical as credulous, rules that are not satisfied (as for choice logic programs) must be (c-)defeated.

Definition 3.6. *Let P be an OCLP. A total interpretation I is a **skeptical/credulous model** iff every rule in $P^\star$ is either not applicable, applied or (c-)defeated w.r.t. I. A skeptical/credulous model M is **minimal** iff M is minimal according to set inclusion, i.e. no skeptical/credulous model N of P exists such that $N \subset M$.*

Example 3.7. Reconsider the interpretations I, J, K and L from Example 3.2. Only K and L are skeptical/credulous models. Model L is not minimal due to the skeptical/credulous model $Z = \{dvd\_player, cd\_writer, laptop, larger\}$. The minimal skeptical/credulous models K and Z correspond to the intuitive outcomes of the problem.

Example 3.8. The program of Example 3.5 has no skeptical models but two credulous ones: $\{a\}$, $\{b\}$.

The next example illustrates that the skeptical/credulous model semantics does not always provide the appropriate solutions to the decision problem at hand.

Example 3.9. Consider the ordered choice logic program $P = \langle \{P_1 = \{a \leftarrow\}, P_2 = \{b \leftarrow\}, P_3 = \{a \oplus b \leftarrow c\}, P_3 \prec P_2 \prec P_1\rangle$, where P has two minimal skeptical/credulous models: $M = \{b, c\}$, and $N = \{a, b\}$. Clearly, c is an unsupported assumption in M, causing P_3 to trigger an unwarranted choice between a and b.

We introduce an adaptation of the Gelfond-Lifschitz [23] and reduct ([25]) transformations to filter unintended (minimal) models containing unsupported atoms. This results in the skeptical/credulous answer set semantics.

Definition 3.7. *Let M be a total interpretation for an OCLP P. The **Gelfond-Lifschitz transformation** (resp. **reduct**) for P w.r.t. M, denoted P^M (resp. P_c^M), is the CLP obtained from $P^\star$ by removing all (c-)defeated rules. M is called a **skeptical (resp. credulous) answer set** for P iff M is a minimal model[7] for P^M (resp. P_c^M).*

Although both answer set semantics produce models (skeptical or credulous ones) for the program, they differ in whether they produce minimal ones or not. Just as for answer sets of semi-negative logic programs, we find that skeptical answer sets are minimal skeptical models. For extended disjunctive logic programs, the answer set semantics is not minimal[25]. The same applies for credulous answer sets of ordered choice logic programs, as demonstrated by the following example.

[7] The definition in [16] states a stable model, but since both are identical for CLP, we have opted in this paper to use minimal model instead.

oarieli@mta.ac.il

Example 3.10. Consider the program $P = \langle \{P_1 = \{r_1 : g \leftarrow\}, P_2 = \{r_2 : p \oplus d \leftarrow; r_3 : g \oplus p \leftarrow; r_4 : g \oplus d \leftarrow\}\}, P_2 \prec P_1 \rangle$. Consider $M_1 = \{g\}$ and $M_2 = \{g, d\}$. Clearly, $M_1^+ \subset M_2^+$, while both interpretations are credulous answer sets for P. For M_1, we have that $P_c^{M_1} = \{g \leftarrow; g \oplus d \leftarrow; g \oplus p \leftarrow\}$ for which it can easily be verified that M_1 is a minimal model. The program $P_c^{M_2} = \{p \oplus d \leftarrow ; g \oplus p \leftarrow\}$ has two minimal models: $\{p\}$ and $\{g, d\}$. Note that M_2 is a credulous model because the c-defeater has become c-defeated, i.e. the justification in M_1 for c-defeating $p \oplus d \leftarrow$ has disappeared in M_2.

Non-minimal credulous answer sets appear when the program contains inconsistencies on a decision level: in the above example the following choices have to be made: $\{p, d\}$, $\{g, p\}$ and $\{g, d\}$. Because of the program's construction, one can choose either one or two alternatives and c-defeating will make the choice justifiable.

4 Implementation

For the last five years, answer set programming has gained popularity. One of the main forces behind this is the growing efficiency of answer solvers like smodels ([20]) and dlv ([26]).

In this section, we propose a mapping, for both semantics, to semi-negative logic programs. Since both answer set solvers support this type of programs, this would allow us to implement an OCLP front-end for them. In this section we will present an initial implementation designed to work on top of smodels. So far our efforts have been restricted to the grounded case, in the last part of this section we extend our approach to non-grounded programs.

The skeptical answer set semantics is based on the notion of defeat. If we want to map our formalism to a language which does not support this, we need a way to encode it. This implies anticipating which combinations of rules could be capable of defeating a rule and which ones are not.

The definition of defeating relies strongly on the notion of alternatives: rules can only be defeated by rules containing alternatives of the head atoms. Therefore, anticipating defeaters also implies predicting alternatives. According to Definition 3.3, b is an alternative of a in a component C if one can find an applicable choice rule as preferred as C containing both a and b in the head. This implies that even without an interpretation we can find out which atoms might be or could become alternatives; it only remains to be checked if the rule is applicable or not. These condition-based alternatives are referred to as possible future alternatives and are defined more formally below.

Definition 4.1. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of **possible future alternatives** of a in C, denoted as $\mathcal{A}_C^P(a)$, is defined as $\mathcal{A}_C^P(a) = \{(b, B_r) \mid \exists r \in P \cdot c(r) \preccurlyeq C, a, b \in H_r, a \neq b\}$.*

Example 4.1. Consider the OCLP $P = \langle \{P_1 = \{r_1 : a \leftarrow; r_2 : f \leftarrow\}, P_2 = \{r_3 : a \oplus b \oplus c \leftarrow d; r_4 : a \oplus d \leftarrow f; r_5 : d \oplus c \leftarrow\}, P_2 \prec P_1\} \rangle$. The possible future alternatives of a in P_1 equal $\mathcal{A}_{P_1}^P(a) = \{(b, \{d\}), (c, \{d\}), (d, \{f\})\}$.

oarieli@mta.ac.il

It is easy to see that one can compute the possible future defeaters in one iteration of the program, making the process polynomial.

The next theorem demonstrates that alternatives can be expressed in terms of possible future alternatives.

Theorem 4.1. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P, $a \in \mathcal{B}_P$ and I an interpretation for P. Then, $\Omega^I_C(a) = \{b \mid (b, S) \in \mathcal{A}^P_C(a) \wedge S \subseteq I\}$.*

Having these possible future alternatives allows us to detect possible future defeaters in much the same way as we detect standard defeaters (Definition 3.4). The only extra bit we need is to collect all the conditions on the alternatives. This collection then acts as the condition for the defeating rule.

Definition 4.2. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of **possible future defeaters** of a in C, denoted as $\mathcal{D}^P_C(a)$, is defined as $\mathcal{D}^P_C(a) = \{(r, S) \mid \exists r \in P \cdot c(r) \prec C, \forall b \in H_r \cdot (b, B_b) \in \mathcal{A}^P_C(a), S = \bigcup_{b \in H_r} B_b\}$. The set of **possible future defeaters** of a rule $r \in P$, denoted as $\mathcal{D}^P(r)$, is defined as $\mathcal{D}^P(r) = \{(R, S) \mid S = \bigcup_{a \in H_r} S_a$ such that $(r_a, S_a) \in \mathcal{D}^P_{c(r)}(a), r_a \in R\}$.*

Having the possible future defeaters of an atom in a certain component, we can easily find that combination that can act as a possible future defeater of a rule in a certain component. We simply compute the set of possible future defeaters of each of the head atoms of this rule in this rule's component. The set of all possible permutations of choosing an element from each of these sets give us the possible future defeaters of our rule. In other words, we obtain a number of possible future defeaters of a rule equal to the product of the sizes of the sets of possible future defeaters for each of its head elements.

Example 4.2. When we look back to the program P of Example 4.1, we have that a has a one possible future defeater in P_1 as: $\mathcal{D}^P_{P_1}(a) = \{(r_5, \{d, f\})\}$. In the same component, we have that c has a future defeater $\mathcal{D}^P_{P_1}(c) = \{(r_4, \{d, f\})\}$. All the other atoms in the program do not have any possible future defeaters in any of the relevant components. The rule r_1 is the only rule with possible future defeaters, namely $\mathcal{D}^P(r_1) = \{(\{r_5\}, \{d, f\})\}$.

Computing all possible defeaters of an atom can be done by one pass of the program. Therefore, computing possibly defeaters of all atoms can be done in polynomial time. The possible future defeaters of a rule can be obtained in polynomial time. By their construction, the number of possible future defeaters is polynomial with respect to number of atoms in the program.

Clearly, possible future defeaters can be used for expressing interpretation-dependent defeaters.

Proposition 4.1. *Let P be an OCLP and let I be an interpretation for it. A rule $r \in P^\star$ is defeated w.r.t. I iff $\exists (R, S) \in \mathcal{D}^P(r) \cdot S \subseteq I, B_{r'} \subseteq I, \forall r' \in R$.*

These possible future defeaters are the key to mapping OCLPs to semi-negative logic programs. We are only required to turn the information which makes possible future defeaters into defeaters, i.e. they have to be applicable, into a condition. To make

oarieli@mta.ac.il

this possible, we introduce for each non-constraint rule r in the program two new atoms: d_r and a_r. The former indicates that the rule r is defeated or not, while the truth value of the latter is an indicator of the applicability of the rule.

Definition 4.3. *Let P be an OCLP. Then, the logic program*[8] $P_\neg$ *is defined as follows:*

1. $|H_r| = 0$*:* $r \in P_\neg$
2. $|H_r| \geq 1$*:*
   a) $h \leftarrow B_r, \neg d_r, \neg(H_r \setminus \{h\}) \in P_\neg$*:* $\forall h \in H_r$
   b) $a_r \leftarrow B_r \in P_\neg$
   c) $d_r \leftarrow C \in P_\neg$ *with* $C = S \cup \bigcup_{r' \in R} a_{r'}$ *such that* $(R, S) \in \mathcal{D}^P(r)$.
   d) $\leftarrow h, g, B_r, \neg d_r \in P_\neg$*:* $\forall h, g \in H_r \cdot h \neq g$

Since constraints are not involved in the defeating process, we can simply copy them to the corresponding logic program. For the answer set semantics of ordered choice logic program, we need, among other things, that each applicable, undefeated rule admits exactly one head atom. Rules of type a) and d) make sure that the corresponding rules in the logic program do not violate this property. The rules of type b) indicate which original rules are applicable. The c)-rules are probably the most difficult ones. They express when a rule should or could be considered defeated. If we look at Theorem 4.1, we have a mechanism for relating possible future defeaters to actual defeaters. Given a possible future defeater (R, S) for a rule r , we simply have to make sure that all rules in R are applicable and that all atoms in S are true with respect to the current interpretation. With rules of type b), we can express the former using a_r. Combining all of this, we can signal in the transformed program that a rule is defeated or not using a rule $d_r \leftarrow a_{r_1}, \ldots, a_{r_n}, S$ with $r_i \in R$ and $n = |H_r|$. Whenever an answer set of the transformed program makes d_r true, we know that the original rule r is defeated. The construction with rules of type b) makes sure that the reverse also holds.

Example 4.3. The corresponding logic program $P_\neg$ of the OCLP of Example 4.1 looks like:

$a \leftarrow \neg d_{r_1}$	$a \leftarrow f, \neg d, \neg d_{r_4}$	$a_{r_2} \leftarrow$	$\leftarrow d, \neg d_{r_3}, a, b$
$f \leftarrow \neg d_{r_2}$	$d \leftarrow f, \neg a, \neg d_{r_4}$	$a_{r_3} \leftarrow d$	$\leftarrow d, \neg d_{r_3}, a, c$
$a \leftarrow d, \neg b, \neg c, \neg d_{r_3}$	$d \leftarrow \neg c, \neg d_{r_5}$	$a_{r_4} \leftarrow f$	$\leftarrow d, \neg d_{r_3}, b, c$
$b \leftarrow d, \neg a, \neg c, \neg d_{r_3}$	$c \leftarrow \neg d, \neg d_{r_5}$	$a_{r_5} \leftarrow$	$\leftarrow f, \neg d_{r_4}, a, d$
$c \leftarrow d, \neg a, \neg b, \neg d_{r_3}$	$a_{r_1} \leftarrow$	$d_{r_1} \leftarrow a_{r_5}, d, f$	$\leftarrow \neg d_{r_5}, d, c$

The original OCLP of Example 4.1 has two skeptical answer sets, $\{f, d, b\}$ and $\{f, c, a\}$, which correspond exactly with the two answer sets, $\{a_{r_1}, a_{r_2}, a_{r_3}, a_{r_4}, a_{r_5}, d_{r_1}, f, d, b\}$ and $\{a_{r_1}, a_{r_2}, a_{r_4}, a_{r_5}, f, c, a\}$, of $P_\neg$.

Because the transformation is based on possible future defeaters, it is easy to see that the transformation can be constructed in polynomial time, leading to only a polynomial increase of rules.

Theorem 4.2. *Let P be an OCLP and $P_\neg$ be its corresponding logic program. Then, a one-to-one mapping exists between the skeptical answer sets M of P and the answer sets N of $P_\neg$ in such a way that $N = M \cup \{a_r \mid \exists r \in P \cdot |H_r| \geq 1, B_r \subseteq M\} \cup \{d_r \mid \exists r \in P \cdot r$ is defeated w.r.t. $M\}$.*

[8] In this paper, we use $\neg$ to represent negation as failure.

oarieli@mta.ac.il

4.1 Credulous Mapping

To obtain the credulous answer set semantics for OCLPs, we propose a similar mapping to semi-negative logic programs. The only difference between the skeptical and the credulous semantics is the way they both handle defeat. For the credulous version, we need to make sure that we look for c-defeaters in all components which are not less preferred as the rule we wish to defeat. Furthermore, we have to make sure that c-defeaters are applied and not just applicable as is the case for defeaters. The former will be encoded by means of possible future c-defeaters while the latter will be translated in a different style of a_r rules in the mapping.

The definition of possible future c-defeater is identical to the one of its skeptical counter-part except that it looks for rules in all components which are not less preferred.

Definition 4.4. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of **possible future c-defeaters** of a in C, denoted as $\mathcal{F}_C^P(a)$, is defined as $\mathcal{F}_C^P(a) = \{(r,S) \mid \exists r \in P \cdot C \not< c(r), \forall b \in H_r \cdot (b,B_b) \in \mathcal{A}_C^P(a), S = \bigcup B_b\}$. The set of **possible future c-defeaters** of a rule $r \in P$, denoted as $\mathcal{F}^P(r)$, is defined as $\mathcal{F}^P(r) = \{(R,S) \mid S = \bigcup_{a \in H_r} S_a$ such that $(r_a, S_a) \in \mathcal{F}_{c(r)}^P(a), r_a \in R\}$.*

Just as their sceptical counterparts, possible future c-defeaters can be obtained in polynomial time. Their number is also polynomial w.r.t. the number of atoms in the program.

Just as before, c-defeaters can be expressed in terms of possible future c-defeaters.

Theorem 4.3. *Let P be an OCLP and let I be an interpretation for it. A rule $r \in P^\star$ is c-defeated w.r.t. I iff $\exists (R,S) \in \mathcal{F}_{c(r)}^P(a) \cdot S \subseteq I, r'$ applied w.r.t. $I, \forall r' \in R$.*

Definition 4.5. *Let P be an OCLP. Then, the logic program $P_\neg^c$ is defined as follows:*

1. $|H_r| = 0$*:* $r \in P_\neg^c$
2. $|H_r| \geq 1$*:*
   a) $h \leftarrow B_r, \neg d_r, \neg(H_r \setminus \{h\}) \in P_\neg^c$*:* $\forall h \in H_r$
   b) $a_r \leftarrow B_r, h, \neg(H_r \setminus \{h\}) \in P_\neg^c$*:* $\forall a \in H_r$
   c) $d_r \leftarrow C \in P_\neg^c$ *with* $C = S \cup \bigcup_{r' \in R} a_{r'}$ *with* $(R,S) \in \mathcal{F}^P(r)$.

The credulous mapping is very similar to the skeptical one but there are a couple of subtle differences: an obvious difference is the use of possible future c-defeater instead of their skeptical counterparts (c-rules). The second change are the rules implying a_r (b-rules). Previously they were used to indicate applicability, the necessary condition for the defeat. Since c-defeat works with applied defeaters, we need to make sure that a_r is considered only true when r is applied. The less obvious change is the absence of the rules of type d). Since a rule can only be applied when one and only one head atom is considered true and because a_r should only be considered true in this particular case, they no longer necessary.

This transformation can also be performed in polynomial time and results in a polynomial increase of rules.

oarieli@mta.ac.il

Example 4.4. Reconsider the OCLP from Example 3.10. If we use the mapping from Definition 4.5, we obtain the following program:

$$
\begin{array}{lll}
g \leftarrow \neg d_1 & a_1 \leftarrow g & d_1 \leftarrow a_2 \\
p \leftarrow \neg d, \neg d_2 & a_2 \leftarrow p, \neg d & d_2 \leftarrow a_3, a_4 \\
d \leftarrow \neg p, \neg d_2 & a_2 \leftarrow d, \neg p & d_3 \leftarrow a_2, a_4 \\
g \leftarrow \neg p, \neg d_3 & a_3 \leftarrow g, \neg p & d_4 \leftarrow a_2, a_3 \\
p \leftarrow \neg g, \neg d_3 & a_3 \leftarrow p, \neg g & \\
g \leftarrow \neg d, \neg d_4 & a_4 \leftarrow g, \neg d & \\
d \leftarrow \neg g, \neg d_4 & a_4 \leftarrow d, \neg g &
\end{array}
$$

The answer sets of this program correspond perfectly to the credulous answer sets of the original program. The newly introduced atoms make sure that the answer set semantics remains minimal while the credulous OCLP version is clearly not.

Theorem 4.4. *Let P be an OCLP and $P_\neg$ be its corresponding logic program. Then, a one-to-one mapping exists between the credulous answer sets M of P and the answer sets N of $P_\neg$ in such a way that $N = M \cup \{a_r \mid \exists r \in P \cdot |H_r| \geq 1, B_r \subseteq M, |H_r \cap M| = 1\} \cup \{d_r \mid \exists r \in P \cdot r$ is c-defeated w.r.t. $M\}$.*

4.2 Using Answer Set Solvers

The transformations presented above give us the theoretical certainty that OCLP can be implemented on top of answer set solvers like Smodels ([20]), and dlv ([26]).

However, no attempt is made to make the mappings efficient, apart from assuring that they can be done in polynomial time. When actually implementing the system, we should take efficiency, both time as space wise, into account. New atoms and rules should only be created when necessary.

For example, if one of the head atoms of a rule does not have any alternatives relative to the component of the rule, there is hardly any point of mentioning d_r, as no rule will be generated anyway. The same applies for the a_r rules: if a rule does not stand the chance to be a (c-)defeater, then there is no point in defining it. Theoretically speaking, there is no need to introduce the atoms a_r. One can easily incorporate the bodies into the rules describing the defeating conditions. Unfortunately, this makes the mapping harder to read and would, in the credulous case, create more rules of type d). For each defeater, one has to create rules to accommodate all the various ways this rule can be made applied. By using a_r, one only needs to do this once, while without it one has to do this over and over again. For larger and more complex programs this can cause a serious overhead. In algorithmic form of the mappings, it would be best to start with generating the c)-rules. By doing so, one can immediate tell which a_r rules are necessary and whether $not d_r$ should be included in the corresponding rule or not.

An other time/space saving point is the effective usage of the various constructs provided by the answer set solver at hand. The disjunctive rules used by dlv can help us to reduce the number of rules, by simply replacing the a)-rules with a single disjunctive rule. Unfortunately, this measure does not eliminate the need for the constraints, assuring

oarieli@mta.ac.il

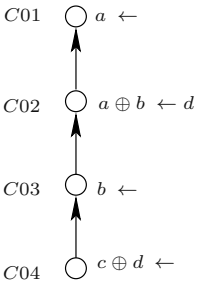


Fig. 2. The OCT program of Example 4.6

that each applicable rule is made applied by a single true head atom. Smodels on the other hand provides us with a mechanism of writing all shifted rules and the corresponding constraints as a single rule using their special choice construct.

Example 4.5. Reconsider the OCLP mentioned in Example 4.1. If we only take into account the special construct provided my smodels, we obtain the following program:

```
a :- not dr1.                    ar2.
f :- not dr2.                    ar3 :- 1{a,b,c}1.
1{a,b,c}1 :- d, not dr3.         ar4 :- 1{a,d}1.
1{a,d}1 :- f, not dr4.           ar5 :- 1{d,c}1.
1{d,c}1 :- not dr5.              dr1 :- ar5, d, f.
ar1.
```

By taken all the other rule and atom saving measures into account, we obtain:

```
a :- not dr1.                    1{d,c}1
f.                               ar5 :- 1{d,c}1.
1{a,b,c}1 :- d.                  dr1 :- ar5, d, f.
1{a,d}1 :- f
```

4.3 Implementation an OCLP Front-End to Smodels

To demonstrate the theoretical mapping and to serve as a basis for future experiments and research a simple language was developed to allow OCLP to be processed by computer. A compiler[9] was created to parse the input language and interface into the Smodels([20]) API which was then used to compute the answer sets. The compiler *OCT* is available under the GPL ("open source") from `http://www.cs.bath.ac.uk/$\sim$mdv/oct/`.

Example 4.6. The OCLP from Figure 2 can be very easily written as an input file for the oct-compiler:

[9] Here compiler is used in the broader sense of an automated computer language translation system rather than traditional procedural to machine code system.

oarieli@mta.ac.il

```
component c01 {  a.  }

component c02 {  a + b :- d. }

component c03 {  b.  }

component c04 {  d + c. }

c04 < c03 < c02 < c01
```

Definitions 10 and 12 give us a theoretical basis for a program to convert OCLPs in semi-negative logic programs but a few changes and optimisations are necessary before we have an effective algorithm for converting and solving OCLPs. All optimisations available in oct are based on the information obtained from the mapping and none use any special constructs provided by Smodels. This allows for the usage of OCT on top of any other answer set solver. Two types of optimisations are provided: intra-transform and inter-transform. The former are carried out during the transformation of OCLP to LP on a one rule basis, the latter are recursively applied between rules after the initial mapping. More information on the various information techniques can be found in [5].

Example 4.7. If we reconsider the OCLP from Example 4.6, OCT without optimisations provides us with the following semi-negative logic program:

```
0 | oclp_a_0 .
  | a :- not oclp_d_0 .
  | oclp_d_0 :- d , oclp_a_2 .
1 | oclp_a_1 :- d .
  | a :- not oclp_d_1 , not b , d .
  | :- not oclp_d_1 , a , b , d .
  | b :- not oclp_d_1 , not a , d .
2 | oclp_a_2 .
  | b :- not oclp_d_2 .
3 | oclp_a_3 .
  | d :- not oclp_d_3 , not c .
  | :- not oclp_d_3 , d , c .
  | c :- not oclp_d_3 , not d .
```

When we allow for optimisations, we obtain a much smaller and compacter program:

```
a :- not d .
:- a , d .
b .
d :- not c .
:- d , c .
c :- not d .
```

oarieli@mta.ac.il

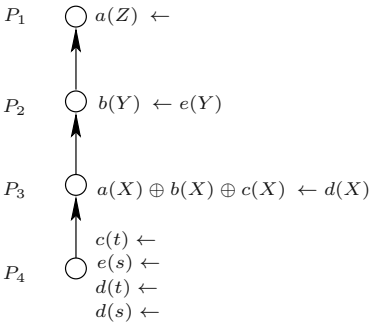


Fig. 3. The non-grounded OCLP of Example 5.1

5 The Non-grounded Case

So far we have always assumed that all our programs were fully grounded, i.e. all rules have been replaced by their ground instances obtained from substituting all variables with constants appearing in the program. This process of grounding has no influence on the semantics of our programs. Using variables makes the program more compact and readable. All our definitions and theorems remain valid in the presence of variables and a countable Herbrand Universe. However, writing them down in a reader-friendly manner becomes more complicated since at various stages variables have to be renamed to avoid confusion and/or incorrect grounding in future stages. Furthermore, the atoms a_r and d_r need to be equipped with variables to make sure that only particular groundings are considered for applicability and (c-)defeat.

Let us demonstrate our approach with an example.

Example 5.1. Consider the non-grounded OCLP in Figure 3. Clearly in this example, $a(X)$, $b(Y)$ and $c(Z)$ can only be alternatives if $d(X)$ is true for some X and if they all have the same substitution. This will be taken into account when obtaining possible future alternatives and possible future (c-)defeaters. This is also reflected in the variables we add to a_r and d_r. For our program, we obtain:

$a(Z) \leftarrow \neg d_{r_1}(Z)$
$b(Y) \leftarrow \neg d_{r_2}(Z)$
$a(X) \leftarrow d(X), \neg d_{r_3}(X), \neg b(X), \neg c(X)$
$b(X) \leftarrow d(X), \neg d_{r_3}(X), \neg a(X), \neg c(X)$
$c(X) \leftarrow d(X), \neg d_{r_3}(X), \neg a(X), \neg b(X)$
$c(t) \leftarrow \neg d_{r_4}$
$e(s) \leftarrow \neg d_{r_5}$
$d(t) \leftarrow \neg d_{r_6}$
$d(s) \leftarrow \neg d_{r_7}$
$a_{r_1} \leftarrow$
$a_{r_2}(Y) \leftarrow e(Y)$
$a_{r_3}(X) \leftarrow$
$a_{r_4} \leftarrow$
$a_{r_5} \leftarrow$
$a_{r_6} \leftarrow$
$a_{r_7} \leftarrow$
$d_{r_1}(Z) \leftarrow d(Z), a_(r_2)(Z)$
$d_{r_1}(t) \leftarrow d(t), a_{r_4}$
$d_{r_2}(t) \leftarrow d(t), a_{r_4}$
$\leftarrow a(X), b(X), \neg d_{r_3}(X), d(X)$
$\leftarrow a(X), c(X), \neg d_{r_3}(X), d(X)$
$\leftarrow b(X), c(X), \neg d_{r_3}(X), d(X)$

oarieli@mta.ac.il

6 Complexity

In this section we investigate the complexity and expressive power of our system. We assume that the reader is familiar with the basic notions and definitions in this area. For a succinct but detailed overview we refer to [4].

In [16], it was shown that ordered choice logic programming is capable of representing extended generalised logic programs (EGLP)[10] while maintaining the answer set semantics as the skeptical answer semantics of the corresponding OCLP. One can prove that the same applies for the credulous answer set semantics, using the same transformation. For EGLP it can be shown that classical negation can easily be removed with the use of a single pre-processor. Therefore, we will assume that all programs are free of classical negation.

Definition 6.1. *Let P be an EGLP without classical negation.. The corresponding OCLP is defined by $\langle\{C, R, N\}, C \prec R \prec N\rangle$ with*

$$
\begin{aligned}
N &= \{\mathrm{not}_a \leftarrow \ \mid a \in \mathcal{B}_P\}\ , \\
R &= \{a \leftarrow B, \mathrm{not}_C \in R \mid r : a \leftarrow B, \neg C \in P\} \cup \\
&\quad \{\mathrm{not}_a \leftarrow B, \mathrm{not}_C \in R \mid r : \neg a \leftarrow B, \neg C \in P\} \cup \\
&\quad \{\ \leftarrow B, \mathrm{not}_C \in R \mid r :\ \leftarrow B, \neg C \in P\}\ , \\
C &= \{a \oplus \mathrm{not}_a \leftarrow a \mid a \in \mathcal{B}_P\}\ ,
\end{aligned}
$$

where, for $a \in \mathcal{B}_P$, not_a is a fresh atom[11] *representing $\neg a$.*

Intuitively, the choice rules in C force a choice between a and $\neg a$ while the rules in N encode "negation by default".

Theorem 6.1. *Let P be an EGLP without classical negation Then, $M \subseteq \mathcal{B}_P$ is an answer set of P iff S is a skeptical/credulous answer set of P_L with $S = M \cup \mathrm{not}_{(\mathcal{B}_P \setminus M)}$.*

Combining this result with the transformations from the previous section, we obtain a mechanism to go, at any time, from a logic program to an ordered choice logic programs and back without changing the semantics of our programs. All three mappings involved in this process can be executed in polynomial time, which implies that semi-negative and ordered choice logic programming have the same complexity and expressive power as far as their (skeptical/credulous) answer set semantics is concerned. A good overview of complexity and expressive power of logic programming can be found in [14].

Theorem 6.2. *Let P be an ordered choice logic program:*

- *Checking if an interpretation M is a skeptical/credulous answer set of P is **P**-complete.*
- *Deciding if P has a skeptical/credulous answer set is **NP**-complete.*

[10] Extended generalised logic programs allow both types of negation (classical and as failure) to appear both in the head and body of a rule. negation as failure in the head of rule. See [25] for more information details.

[11] For a set $X \in \mathcal{B}_P$, $\mathrm{not}_X = \{\mathrm{not}_a \mid a \in X\}$.

oarieli@mta.ac.il

- *Ordered choice logic programming under the skeptical/credulous answer set semantics is co-**NP**-complete.*
- *If we allow function symbols with non-zero arity, the complexity of OCLP becomes: $\mathbf{\Pi}^1_1$-complete.*
- *Full OCLP under the skeptical/credulous semantics expresses or captures $\mathbf{\Pi}^1_1$.*

7 Relationship to Other Approaches

Various logic (programming) formalisms have been introduced to deal with the notions of preference, order and updates. Ordered choice logic programming uses the intuition of defeating from ordered logic programming (OLP) [22,24] to select the most favourable alternative of a decision. In fact, every ordered logic program can be transformed into a OCLP such that the answer set semantics reflects the credulous semantics of the OCLP.

Dynamic preference in extended logic programs was introduced in [8] in order to obtain a better suited well-founded semantics. Although preferences are called dynamic they are not dynamic in our sense. Instead of defining a preference relation on subsets of rules, preferences are incorporated as rules in the program. Moreover, a stability criterion may come into play to overrule preference information. Another important difference with our approach is the notion of alternatives, as the corresponding notion in [8] is statically defined.

A totally different approach is proposed in [28]. where preferences are defined between atoms. Given these preferences, one can combine them to obtain preferences for sets of atoms. Defining models in the usual way, the preferences are then used to filter out the less preferred models.

[11] proposes disjunctive ordered logic programs which are similar to ordered logic programs [22] where disjunctive rules are permitted. In [10], preference in extended disjunctive logic programming is considered. As far as overriding is concerned the technique corresponds to a skeptical version of the OCLP semantics ([16]), but alternatives are fixed as an atom and its (classical) negation.

To reason about updates of generalised logic programs, extended logic programs without classical negation, [2] introduces dynamic logic programs. A stable model of such a dynamic logic program is a stable model of the generalised program obtained by removing the rejected rules. The definition of a rejected rule corresponds to our definition of a defeated rule when a and $\neg a$ are considered alternatives. It was shown in [2], that the stable model semantics and the answer set semantics coincide for generalised logic programs. In Theorem 6.1 we have demonstrated that extended logic programs without classical negation can be represented as ordered choice logic programs such that the answer set semantics of the extended logic program can be obtained as the answer set semantics of the OCLP. Because rejecting rules corresponds to defeating rules, it is not hard to see that, with some minor changes, Definition 6.1 can be used to retrieve the stable models of the dynamic logic program as the stable models of the corresponding OCLP. The only things we need to do are to replace the component R by the P_is of the dynamic logic program $\bigoplus\{P_i : i \in S\}$, replace every occurrence of $\neg a$ by not_a and add $a \oplus \mathrm{not}_a \leftarrow \mathrm{not}_a$ to C for each $a \in \mathcal{B}_P$.

oarieli@mta.ac.il

A similar system is proposed in [19], where sequences are based on extended logic programs, and defeat is restricted to rules with opposing heads. The semantics is obtained by mapping to a single extended logic program containing expanded rules such that defeated rules become blocked in the interpretation of the "flattened" program.

A slightly different version of Definition 6.1 can be used to map the sequences of programs of [19] to OCLPs.

In [3], preferences are added to the dynamic logic program formalism of [2]. These are used to select the most preferred stable models. [29] also proposes a formalism that uses the order among rules to induce an order on answer sets for inconsistent programs, making it unclear on how to represent decisions. Along the same line, [18] proposes logic programs with compiled preferences, where preferences may appear in any part of the rules. For the semantics, [18] maps the program to an extended logic program.

[6,9,7] use yet an other approach. They define preferences between the head atoms of a disjunction. The first head atom is more preferred than the second which is more preferred that the third, etc. So in a sense one can say that these atoms become alternatives. In our approach we define the preferences between the alternatives in separate rules which each conditions of their own. To our opinion this allows for a greater sense of freedom for the programmer. Furthermore, in our system, it is very easy to adapt to changes over time, as decisions can be overruled or defeated. However, one thing has definitely to be said for their approach, they can represent problems in Σ_2^P while our approach is restricted to Σ_1^P. This higher order of complexity is achieved by allowing multiple alternatives to be decided upon without overruling the decision.

8 Conclusions and Directions for Future Research

In this paper we proposed a mechanism for transforming ordered choice logic programs to semi-negative logic program while preserving, depending on the transformation, the skeptical or credulous answer set semantics. We examined the possible ways in which the proposed theoretical mappings could be made more efficient, when used on top of an answer set solver. On the more theoretical side, these transformations allowed us to study the complexity and the expressiveness of our formalism.

Previously, OCLP was used to describe and to reason about game theory ([16,17]). It was shown that OCLP is an elegant and intuitive mechanism for representing extensive games with perfect information. The Nash and subgame perfect equilibria can easily be obtained as the credulous answer sets of the corresponding programs. To this extent, we used a special class of OCLPs. Combining these special characteristics with the mapping of OCLP to logic programs, we can create a game-theory tailored front-end to answer set solvers.

In [17], we proposed a multi-agent system where the knowledge and beliefs of the agents is modelled by an OCLP. The agents communicate with each other by sending skeptical/credulous answer sets. The notion of evolutionary fixpoint shows how the various agents reason in order to come to their final conclusions. Having an implementation for OCLP would allow us to implement multi-agent systems and experiment with them in various domains. One possibility would be incorporating this technology into Carel ([30]), a multi-agent system for organ and tissue exchange.

oarieli@mta.ac.il

In the same domain of multi-agent systems it would be interesting to find out if, with this technique, we could include preferences in the DALI-system ([13]). Future research will also focus on using OCLP as a tool in multi-agent institutions [27] to enforce norms and regulations [21] between participating agents.

References

1. *Logic in Artificial Intelligence*, volume 1919 of *Lecture Notes in Artificial Intelligence*, Cosenza, Italy, September 2002. Springer Verlag.
2. José Júlio Alferes, Leite J. A., Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In Cohn et al. [12], pages 98–111.
3. José Júlio Alferes and Luís Moniz Pereira. Updates plus preferences. In *European Workshop, JELIA 2000*, volume 1919 of *Lecture Notes in Artificial Intelligence*, pages 345–360, Malaga, Spain, September–October 2000. Springer Verslag.
4. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
5. Martin Brain and Marina De Vos. Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice. In *ASP03: Answer Set Programming: Advances in Theory and Implementation*. Ceur-WS, 2003. online CEUR-WS.org/Vol-78/asp03-final-brain.ps.
6. G. Brewka, Benferhat, and D. Le Berre. Qualitive Choice Logic. In *Principles of Knowledge Representation and Reasoning. KR-02*. Morgan Kaufmann, 2002.
7. G. Brewka, I Niemelä, and M Truszczynski. Answer set programming. In *International Joint Conference on Artificial Intelligence (JCAI 2003)*. Morgan Kaufmann, 2003.
8. Gerhard Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.
9. Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *European Workshop, JELIA 2002* [1].
10. Francesco Buccafurri, Wolfgang Faber, and Nicola Leone. Disjunctive Logic Programs with Inheritance. In Danny De Schreye, editor, *International Conference on Logic Programming (ICLP)*, pages 79–93, Las Cruces, New Mexico, USA, 1999. The MIT Press.
11. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Disjunctive ordered logic: Semantics and expressiveness. In Cohn et al. [12], pages 418–431.
12. Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, Trento, June 1998. Morgan Kaufmann.
13. Stefania Costantini and Arianna Tocchio. A Logic programming Language for Multi-Agent Systems. In *Logics in Artificial Intelligence, Jelia2002*, number 2424 in Lecture Notes in Artificial Intelligence, 2002.
14. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425, 2001.
15. Marina De Vos and Dirk Vermeir. On the Role of Negation in Choice Logic Programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 236–246, El Paso, Texas, USA, 1999. Springer Verslag.
16. Marina De Vos and Dirk Vermeir. Dynamic Decision Making in Logic Programming and Game Theory. In *AI2002: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 36–47. Springer, December 2002.

oarieli@mta.ac.il

17. Marina De Vos and Dirk Vermeir. Logic Programming Agents Playing Games. In *Research and Development in Intelligent Systems XIX (ES2002)*, BCS Conference Series, pages 323–336. Springer, December 2002.
18. J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In W. Horn, editor, *European Conference on Artificial Intelligence*, pages 392–398, 2000.
19. Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On Properties of update Sequences Based on Causal Rejection. *Theory and Practice of Logic Programming*, 2(6), November 2002.
20. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
21. Marc Esteva, Julian Padget, and Carles Sierra. Formalizing a language for institutions and norms. In Jean-Jules Meyer and Milinde Tambe, editors, *Intelligent Agents VIII*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 348–366. Springer Verlag, 2001. ISBN 3-540-43858-0.
22. D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217, Cambridge, Mass, 1991. Morgan Kaufmann.
23. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.
24. Els Laenens and Dirk Vermeir. A Universal Fixpoint Semantics for Ordered Logic. *Computers and Artificial Intelligence*, 19(3), 2000.
25. Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
26. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
27. Joan-Antoní Rodríguez, Pablo Noriega, Carles Sierra, and Julian Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, April 1997. ISBN 0-9525554-6-8.
28. Chiaki Sakama and Katsumi Inoue. Representing Priorities in Logic Programs. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 82–96, Cambridge, September2–6 1996. MIT Press.
29. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. In *European Workshop, JELIA 2002* [1], pages 432–443.
30. Javier Vázquez-Salceda, Julian Padget, Ulises Cortés, Antonio López-Navidad, and Francisco Caballero. Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, 27(3):233–258, 2003. published by Elsevier.

oarieli@mta.ac.il

Skyline Cardinality for Relational Processing

How Many Vectors Are Maximal?

Parke Godfrey

York University, Toronto, Ontario M3J 1P3, Canada
godfrey@cs.yorku.ca

Abstract. The *skyline clause*—also called the *Pareto clause*—recently has been proposed as an extension to SQL. It selects the tuples that are Pareto optimal with respect to a set of designated skyline attributes. This is the *maximal vector problem* in a relational context, but it represents a powerful extension to SQL which allows for the natural expression of on-line analytic processing (OLAP) queries and preferences in queries.
Cardinality estimation of skyline sets is the focus in this work. A better understanding of skyline cardinality—and other properties of the skyline—is useful for better design of skyline algorithms, is necessary to extend a query optimizer's cost model to accommodate skyline queries, and helps to understand better how to use skyline effectively for OLAP and preference queries.
Within a basic model with assumptions of *sparseness* of values on attributes' domains and statistical independence across attributes, we establish the expected skyline cardinality for skyline queries. While asymptotic bounds have been previously established, they are not widely known nor applied in skyline work. We show concrete estimates, as would be needed in a cost model, and consider the nature of the distribution of skyline. We next establish the effects on skyline cardinality as the constraints on our basic model are relaxed. Some of the results are quite counter-intuitive, and understanding these is critical to skyline's use in OLAP and preference queries. We consider when attributes' values repeat on their domains, and show the number of skyline is diminished. We consider the effects of having Zipfian distributions on the attributes' domains, and generalize the expectation for other distributions. Last, we consider the ramifications of correlation across the attributes.

1 Introduction

To query relational data to find a best match, the aggregation operators min and max allow one to retrieve the best—that is, either lowest or highest valued—tuples with respect to a single criterion. The order by clause in SQL allows one to rank order the results, perhaps with respect to many criteria.[1] Beyond this, relational query languages as SQL provide little else for finding best matches, or for expressing preferences as part of one's queries.

[1] The rank ordering will be equivalent to a nested sort over the indicated attributes' values.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 78–97, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

Consider a table of restaurant guide information, as in Figure 1(a). Column **S** stands for *service*, **F** for *food*, and **D** for *decor*. Each is scored from 1 to 30, with 30 as the best.[2] We are interested in choosing a restaurant from the guide, and are looking for a best choice, or a set of best choices from which to choose. Ideally, we would like the restaurant chosen to be the best for service, food, *and* decor, *and* be the lowest priced. There is likely no restaurant that is better than *all* others on *all* criteria, however.[3] No one restaurant trumps all others. For instance, Summer Moon is best on food, but Zakopane is best on service.

restaurant	**S**	**F**	**D**	**price**
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Brearton Grill	15	18	20	62.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50
Briar Patch BBQ	14	13	3	22.50

(a) Restaurant guide table, GoodEats.

restaurant	**S**	**F**	**D**	**price**
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50

(b) The skyline.

Fig. 1. The restaurant table and the skyline.

While there is no one best restaurant, we can eliminate from consideration any restaurant that is worse on all the criteria than another. The Briar Patch BBQ should be eliminated because the Fenton & Pickle is better in comparison across all our criteria. The Brearton Grill is eliminated, in turn, because Zakopane is better than it on all criteria. If Zakopane were not in the table, the Brearton Grill would have remained a consideration. Meanwhile the Fenton & Pickle is worse on every criterion than every other (remaining) restaurant, except on price, where it is the best. So it stays in consideration. (If we were to remove price as one of our criteria, the Fenton & Pickle would be eliminated too.) This would result in the choices in Figure 1(b).

In [2], an extension to SQL is proposed, the skyline of clause, which allows easy expression of the restaurant query we imagined above. They propose also the *skyline operator* as the relational algebraic counterpart of the clause, and pursue an implementation to support efficiently skyline queries within a relational environment.

The skyline of clause is shown in Figure 2(a). It is syntactically similar to an order by clause. Columns $a_1, \ldots, a_n$ are the attributes over which our preferences apply. Their domains must have a natural total ordering, such as integers, floats, and dates. The directives min and max specify whether we prefer low or high values, respectively. The directive diff states that one wants to retain best choices

[2] This table is modeled on the Zagat Survey Guides. For example, see [1].
[3] This is certainly the case in real life, and in real data!

oarieli@mta.ac.il

```
select ... from ... where ...
    group by ... having ...
    skyline of a1 [min | max | diff], ...,
               an [min | max | diff]
```

(a) Skyline clause for SQL.

```
select * from GoodEats
    skyline of S max, F max,
               D max, price min
```

(b) Choosing restaurants.

Fig. 2. Skyline queries.

with respect to each distinct value of that attribute. Let min be the default directive, if none is stated. The skyline query in Figure 2(b) over the table GoodEats in Figure 1(a) expresses what we had in mind above for choosing "best" restaurants, and would result in the answer set in Figure 1(b). If the table GoodEats had a column **C** for *cuisine*, we could add C diff to the skyline of clause to find the best restaurants by each cuisine group.

```
select c1, ..., ck, s1, ..., sm, d1, ..., dn
    from OurTable
except
select D.c1, ..., D.ck, D.s1, ..., D.sm,
      D.d1, ..., D.dn
    from OurTable T, OurTable D
    where D.s1≥ T.s1 and ... D.sm≥ T.sm and
        (D.s1> T.s1 or ... D.sm> T.sm) and
        D.d1= T.d1 and ... D.dn= T.dn
```

Fig. 3. SQL for generating the skyline set.

Skyline queries are not outside the expressive power of present SQL. The query in Figure 3 demonstrates one way to write an arbitrary skyline query in present SQL. The c_i's are attributes of OurTable that we are interested to retain in our query, but that are not skyline criteria. The s_i are the skyline attributes to be minimized, and would appear in skyline of as s_i min. (Without loss of generality, we only consider min and not max.) The d_i are the attributes that are the skyline criteria to *differ*, and would appear in skyline of as s_i diff. It is cumbersome to write skyline-like queries currently in SQL, however. The skyline clause would be a useful syntactic addition to SQL, therefore, if skyline-like queries were to become commonplace. More important than ease of expression, however, is the expense of evaluation. The query in Figure 3 can be prohibitively expensive. It involves a self-join over a table. This join is a θ-join, not an equality-join. It effectively computes the tuples that are trumped—or *dominated*—by other tuples. The tuples then that remain—that were never dominated—determined by the except operation, constitute the skyline tuples.

oarieli@mta.ac.il

There has been a fair amount of recent work focused on how to compute efficiently skyline queries within relational systems for large datasets. In Section 2, we discuss the related work. Skyline cardinality estimation is the focus of this work. A better understanding of skyline cardinality

- is useful for better design of skyline algorithms,
- is necessary to extend the query optimizer's cost model to accommodate skyline queries, and
- helps us to understand better how to use skyline effectively for preference queries.

In Section 3, we present a *basic model*—with assumptions of sparseness over attributes' domains (namely that there are virtually no duplicate values) and statistical independence across attributes—and devise concrete estimates of skyline query cardinalities under the basic model. We consider next the distribution of the number of skyline. The expected value would not be of much utility if the distribution were not to be well behaved. We show through Monte Carlo simulations the nature of the distribution, and that it is well behaved.

In Section 4, we consider the effects on the estimates and distributions as we relax the assumptions of the basic model, thus modeling better the characteristics of real data. We consider the effects of when attribute domains are restricted to small domains (allowing repeated values and ties), different distributions over attribute domains (namely, Zipfian), and pair-wise correlation between attributes. Under the basic model, the number of skyline tuples is *independent* of the distributions of the attributes' domains. This is no longer necessarily true when attributes' values are no longer sparse over their domains. When there are duplicate values over the attributes, the number of skyline tuples *decreases*. We show that the skyline estimates of the basic model are a *ceiling* for skyline cardinalities when the sparseness condition is violated.[4]

In Section 5, we discuss briefly some of the ramifications of our results for skyline algorithm design, cost models for the skyline operator, and the use of skyline in preference queries, and we conclude. We believe these studies can also offer general insights into queries over multi-dimensional criteria.

2 Related Work

The concept of skyline in itself is not new, of course. The search for optimal solutions is a well-established endeavor with an exceedingly deep literature. Beginning in the 1960's, work focused on optimization with respect to multiple criteria. Techniques have been explored for finding good *utility functions* to combine effectively the multiple criteria of interest into a single score. Then traditional mathematical techniques for finding the optimal solution—with respect to a single criterion, the utility function in this case—could then be applied. Others recognized though that for many applications it is often difficult—if not virtually impossible—to find a reasonable utility function. Thus work in *multiple*

[4] There is an exception case, to be discussed.

oarieli@mta.ac.il

criteria optimization focused on how to find *all* optimal solutions in the space with respect to the multiple criteria [3]. The definition of solution in this context is often identical to our definition for skyline: no other potential solution is better across all the criteria. This is called *Pareto optimal*.

The problem attracted attention early on within the mathematics and statistics communities, and within the computational geometry community, since the Pareto optimal points are a super-set of the points that delineate the convex hull of the entire set. How to compute the convex hull, and its size, is of central interest in linear optimization problems. Perhaps in [4] is the first work to study the distribution of *admissible* (Pareto optimal) points, and the expected value of their number. More recently in [5], the work of [4] is extended to show bounds on the variance of the number of admissible points. In [6], Golin considers the effects of the shape of the space to which the points are restricted, and how that shape changes the expected value of the number of admissible points.

Multiple-criterion optimization usually assumes an implicit solution space from which the optimal solutions are to be found. Often, this space is quite large, but also has properties that help to devise good techniques. For skyline queries, the space is explicit: it is the input relation of vectors, or tuples. One does not know necessarily particular properties of the space.

The skyline idea has been studied before in this context of an explicit solution space as the *maximal vector problem*. In [7], the first algorithm to find the maximal vectors (or skyline tuples) from a set of vectors (or relation) was devised. In [8], the maximal vector problem is addressed in the context of computational geometry. In [9], they established that the average number of skyline tuples is $\mathcal{O}((\ln n)^{d-1})$.[5] This is the cardinality bound most often cited and employed in skyline work. However, this is a loose upper-bound. In [10], it is established that $\Theta((\ln n)^{d-1}/(d-1)!)$. There has been work also to establish theoretical complexity bounds for computing the maxima (the Pareto points) and those points on the convex hull [11,12]. This work does not lead directly to good algorithms in practice, however, for computing skyline queries.

Interest has returned to the maximal vector problem recently indeed in the guise of skyline queries. Previous work was main-memory based though, and not well suited to databases. Progress has been made recently on how to compute efficiently such queries in a relational system and over large datasets [2,13,14, 15,16]. In [2], the skyline operator is introduced. They posed two algorithms for it, a block-nested loops style algorithm (and variations) and a divide-and-conquer approach derived from work in [7,8]. In [14,15,16], algorithms for skyline evaluation that exploit indexes are developed. In [13], we developed a general skyline algorithm, based on the "block-nested loops" algorithm of [2], which is more efficient, pipelinable, and amenable to use in a relational query optimizer.

A related topic is nearest-neighbor search. This has been studied in the context of relational systems too [17]. In [18], elements of a cost model for nearest-neighbor searches are considered, but just for high-dimensional cases. In [19],

[5] For this, they made essentially the same assumptions that we shall make in Definition 3.1 about attributes' distributions and independence.

oarieli@mta.ac.il

a specialized index structure is developed to facilitate nearest-neighbor queries. In [14], they employ nearest-neighbors algorithms to pipeline the generation of skyline tuples.

Interest in skyline queries arises in most part from the desire to support queries with preferences in relational systems. In [20], a more general operator called *winnow* is introduced for the purpose of expressing preference queries. Skyline is a special case of winnow. Skyline and related techniques could make it possible to integrate easily certain *cooperative query answering*, *query relaxation*, and *preference query* techniques which have been proposed [21,22,23,24]. In [25], a framework is presented for how to express and combine effectively preferences in queries. In [26], a system called PREFER is presented which is designed to handle multi-parametric ranked queries efficiently. In [27], a preference algebra is developed along with extensions to SQL for general preference queries. They call for a more efficient means to compute preference queries. In [28], a system that incorporates the preference SQL of [27] is presented.

3 Pareto / Skyline Cardinality

We want to estimate the cardinality of the output relation of the skyline operator based upon its input relation. The input can be a base table of the database or a virtual table which is the intermediate result in a query's evaluation. Let us establish a basic model of assumptions about the input relation under which it is possible to establish analytically the cardinality.

Definition 3.1. Basic model of the input relation and skyline query.

Let dimension *refer to an attribute of the relation that participates in the skyline criteria.*

a. Domain assumption (sparseness): *For each dimension, we assume that there are no duplicate values on the attribute across the tuples of the relation.*

b. Independence assumption: *The dimensions are statistically independent.*[6]

Consider a skyline operation with d dimensions over such an input relation of n tuples. (So the relation has at least d attributes, which obey the assumptions above.) Let $\mathsf{s}_{d,n}$ *be the random variable which measures the number of tuples (the cardinality) of the output relation (that is, the set of the resulting skyline tuples). Let* $\hat{\mathsf{s}}_{d,n}$ *denote the expected value of* $\mathsf{s}_{d,n}$*.*

We are interested to know $\hat{\mathsf{s}}_{d,n}$. Under our basic model, no two input tuples share a value over any dimension. Thus, the tuples can be ordered totally on any given dimension. It is not necessary therefore to consider the actual values of the tuples. Instead, we can conceptually replace the value on, say, dimension i of a tuple by its *rank* in the total ordering along dimension i. Without loss of generality, let us assume that we are *minimizing* over the dimensions for the skyline. Let rank 1 refer to the tuple with the smallest value (on that dimension), and n the one with the largest (n is the number of input tuples). We can now just

[6] That is, there are no pair-wise or group correlations nor anti-correlations.

oarieli@mta.ac.il

refer to a tuple's rank on a dimension and ignore the actual value. We provide the proof as it is illustrative for the following discussion.

Theorem 3.1. *[9] The skyline expected value* $\hat{\mathsf{s}}_{d,n}$ *for* $d > 1$ *and* $n > 0$ *obeys the following recurrence equation.*

$$\hat{\mathsf{s}}_{d,n} = \frac{1}{n}\hat{\mathsf{s}}_{d-1,n} + \hat{\mathsf{s}}_{d,n-1}$$

For $n > 0$, $\hat{\mathsf{s}}_{1,n} = 1$.

Proof. *Consider* $\hat{\mathsf{s}}_{1,n}$. *Since no two tuples share the same value on the dimension, only the tuple with rank* 1 *is in the skyline.*

Consider $\hat{\mathsf{s}}_{d,n}$, *for* $d > 1$. *One tuple has rank* n *on dimension* 1. *This tuple cannot dominate any other tuple, since it has a higher value on dimension* 1 *than any other. What is the probability that this tuple itself is a skyline tuple? It is the probability that no other tuple dominates it on dimensions* $2, \ldots, d$, *given the independence assumption. As* $\hat{\mathsf{s}}_{d-1,n}$ *is the expected value of the number of skyline tuples out of* n *tuples on* $d-1$ *dimensions, then* $\frac{1}{n}\hat{\mathsf{s}}_{d-1,n}$ *represents the probability that this one tuple is part of the skyline.*

Since the n-th *ranked tuple on dimension* 1 *cannot dominate any other tuple, the estimated number of skyline tuples of the remaining* $n-1$ *is* $\hat{\mathsf{s}}_{d,n-1}$. □

The recurrence for $\hat{\mathsf{s}}_{d,n}$ is related to the harmonic numbers.

Definition 3.2. Harmonic numbers.

a. *The* harmonic of n, *for* $n > 0$: $\mathsf{H}_n = \sum_{i=1}^{n} \frac{1}{i}$

b. *[29] The* k-th order harmonic of n, *for integers* $k > 0$ *and integers* $n > 0$: $\mathsf{H}_{k,n} = \sum_{i=1}^{n} \frac{\mathsf{H}_{k-1,i}}{i}$

Define $\mathsf{H}_{0,n} = 1$, *for* $n > 0$. *Define* $\mathsf{H}_{k,0} = 0$, *for* $k > 0$.

c. *The* k-th hyper-harmonic of n, *for integers* $k > 0$ *and integers* $n > 0$: $\mathcal{H}_{k,n} = \sum_{i=1}^{n} \frac{1}{i^k}$

A common two-parameter generalization of the harmonic series is that of the hyper-harmonics in Definition 3.2(c). The $\mathcal{H}_{k,n}$ converge for $k > 1$. A second two-parameter generalization is given in Definition 3.2(b), introduced in [29]. (What is effectively a generalization of the $\mathsf{H}_{k,n}$'s appears in [30], in Section 1.2.9.) The $\mathsf{H}_{k,n}$ do not converge for $k > 1$. Rather, $1 \leq \mathsf{H}_{k,n} < n$, for $k > 0$ and $n > 0$. Furthermore,

$$\hat{\mathsf{s}}_{d+1,n} = \mathsf{H}_{d,n} = \sum_{i_1=1}^{n} \sum_{i_2=1}^{i_1} \cdots \sum_{i_d=1}^{i_{d-1}} \frac{1}{i_1 i_2 \cdots i_d}$$

Any particular $\mathsf{H}_{k,n}$ can be solved in terms of $\mathcal{H}_{j,n}$'s ($1 \leq j \leq k$). The H_n and $\mathcal{H}_{k,n}$ are easy to compute, or approximate, and so could be used within a cost model on-the-fly. (Note that $\mathsf{H}_n = \mathcal{H}_{1,n}$.) For instance, we can derive that

oarieli@mta.ac.il

– $\mathsf{H}_{2,n} = \frac{1}{2}\mathsf{H}_n^2 + \frac{1}{2}\mathcal{H}_{2,n}$,
– $\mathsf{H}_{3,n} = \frac{1}{6}\mathsf{H}_n^3 + \frac{1}{2}\mathsf{H}_n\mathcal{H}_{2,n} + \frac{1}{3}\mathcal{H}_{3,n}$, and
– $\mathsf{H}_{4,n} = \frac{1}{24}\mathsf{H}_n^4 + \frac{1}{3}\mathsf{H}_n\mathcal{H}_{3,n} + \frac{1}{8}\mathcal{H}_{2,n}^2 + \frac{1}{4}\mathsf{H}_n^2\mathcal{H}_{2,n} + \frac{1}{4}\mathcal{H}_{4,n}$.

This can be generalized as follows.

Theorem 3.2. $\mathsf{H}_{k,n} = \sum_{\substack{c_1,\ldots,c_k \geq 0 \ \wedge \\ 1\cdot c_1 + 2\cdot c_2 + \ldots + k\cdot c_k = k}} \prod_{i=1}^{k} \frac{\mathcal{H}_{i,n}^{c_i}}{i^{c_i} \cdot c_i!}$ *for* $k \geq 1$ *and* $n \geq 1$, *with the* c_i*'s as integers.*

Proof. *This follows from Knuth's generalization via generating functions [30].* □

For any k, the coefficients of the terms in the summation sum to one. The number of terms to express $\mathsf{H}_{k,n}$ in terms of $\mathcal{H}_{j,n}$'s is $\wp(k)$, the number of ways to partition the positive integer k as a sum of positive integers. Since $\wp(k)$ grows quickly—for instance, $\wp(10) = 42$ and $\wp(20) = 627$—it is not a viable to solve for $\mathsf{H}_{k,n}$'s in this way.

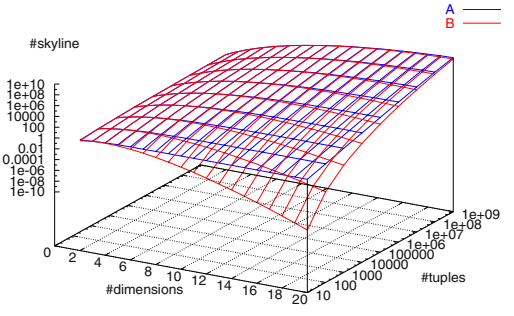


Fig. 4. Plot of $\hat{\mathsf{s}}_{d,n}$ (A) and Buchta's (B).

From this though, we can prove that $\mathsf{H}_{k,n}$ is $\Theta(\mathsf{H}_n^k/k!)$, or equivalently, $\Theta((\ln n)^k/k!)$. In [10], Buchta established that

$$\hat{\mathsf{s}}_{k+1,n} = \frac{(\ln n)^k}{k!} + \frac{\gamma \cdot (\ln n)^{k-1}}{(k-1)!} + \mathcal{O}(\ln n)^{k-2}$$

and therefore that $\hat{\mathsf{s}}_{d,n}$ is $\Theta((\ln n)^{d-1}/(d-1)!)$. We can use these as approximations for $\hat{\mathsf{s}}_{d,n}$. In Figure 4, we plot Buchta's "approximation" (setting the multiplicative constant at one and the additive constant at zero for the $\mathcal{O}$-term) and the actual $\mathsf{H}_{k,n}$ values. $\mathsf{H}_n^k/k!$ and Buchta's underestimate $\mathsf{H}_{k,n}$, with Buchta's

oarieli@mta.ac.il

being marginally closer. Interestingly, while $\mathsf{H}_n^k/k!$ and Buchta's are both monotonically increasing with respect to n, neither is with respect to k. So neither is a good concrete estimate for higher dimensions where the divergence becomes more significant.

Of course, the $\mathsf{H}_{k,n}$ can be numerically approximated off-line, and then a look-up table used at run-time. We computed $\mathsf{H}_{d-1,n}$ for $d = 2, \ldots, 20$ and $n = 10, \ldots, 10^7$ by factors of ten, as appears in Figure 4. As with logarithms, interpolation can be safely used to estimate values that are not in the look-up table.

We have the expected value of skyline cardinality, $\hat{\mathsf{s}}_{d,n}$ (with respect to basic model in Definition 3.1), but we do not know the distribution of the random variable $\mathsf{s}_{d,n}$. If its variance were huge, for instance, our use of $\hat{\mathsf{s}}_{d,n}$ in a cost model for a query optimizer would be of limited utility. It would also be possible for the median to be less than the mean, with a long tail towards n. We are really interested in likely values the optimizer will encounter, not the *expected value*, per se, which itself as a value might never occur. In a cost model, it is important to anticipate the chance of being significantly off the estimation, especially in the case when the actual cardinality is exceedingly larger than the estimate. The query plan can be made to accommodate contingencies, to varying degrees.

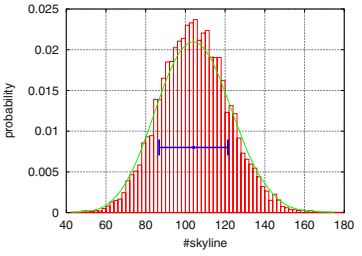
For the case of $d = 2$, it can be proven that the distribution tends to Gaussian by the Central Limit Theorem. (A proof is sketched in [4].) For $d > 2$, the nature of the distribution remains unknown [4]. There is speculation that the distributions for $d > 2$ also tend to Gaussian, and we present experimental support for this, but it remains unproven.

It is possible to infer some properties of the distribution just given what we know of $\hat{\mathsf{s}}_{d,n}$. The domain of $\mathsf{s}_{d,n}$ is $1 \ldots n$, of course. For d and n combinations that are likely for skyline queries in practice, $\hat{\mathsf{s}}_{d,n}$ is close to the 1-end of the spectrum. (See Figure 5(d).) This statistically limits how large the variance can be. (There cannot be much probability that $\mathsf{s}_{d,n}$ is near n, since this would serve to inflate $\hat{\mathsf{s}}_{d,n}$, as by Chebychev's Inequality.)

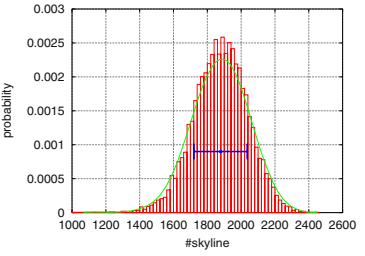
We study experimentally the distribution. We ran Monte Carlo simulations as follows. For each *trial*, we generated one million tuples of d attributes randomly. Each attribute was of type integer and its value was randomly chosen across all values.[7] The number of skyline tuples (minimizing over the d values) was then determined. A simulation then consisted of $10,000$ trials. Figures 5(a), (b), and (c) show the distributions of the $10,000$ trials for $\mathsf{s}_{d,10^6}$ for $d = 3$, 5, and 7, respectively. The data points were binned into about sixty bins in each case to approximate the distribution via a histogram. We normalize the y-axis to probability so the area covered by the histogram is one. In each case, the error-bar represents the mean and spans one standard deviation to the left and to the right of the mean. The super-imposed curve is a bezier fit of the data. The distributions of $\mathsf{s}_{d,n}$ are well behaved and resemble normal (Gaussian) distributions. That the medians are so near the means in the simulations offers evidence that the true distributions have little if no skew.

[7] The values range over 1, ..., 2,147,483,647.

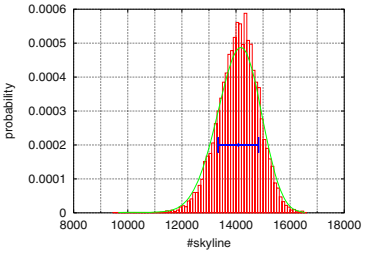
oarieli@mta.ac.il



(a) Distribution of $\mathsf{s}_{3,10^6}$.



(b) Distribution of $\mathsf{s}_{5,10^6}$.



(c) Distribution of $\mathsf{s}_{7,10^6}$.

	$\mathsf{H}_{d-1,10^6}$	Monte Carlo simulations		
d	$\hat{\mathsf{s}}_{d,10^6}$	μ	median	σ
3	104	104	104	17.3
5	1,880	1,878	1,882	158.5
7	14,087	14,087	14,120	745.9

(d) Means, medians, and standard deviations.

Fig. 5. Distributions of $\mathsf{s}_{d,10^6}$.

In [4], they too are interested in the variance of the number of maxima. They establish that the variance of $\mathsf{s}_{d,n}$ converges from above on the expected value, $\hat{\mathsf{s}}_{d,n}$, for any fixed d as n grows. However, this convergence must be extremely slow, as we see. Still, this is further proof that the variance is quite well behaved, and only becomes better behaved as n becomes larger.

4 Generalizing the Basic Model

We explore next the effects of relaxing the assumptions of our basic model from Definition 3.1.

First, we consider the ramifications of relaxing the sparseness condition. Thus tuples may now match on values, and in the extreme, two tuples may be equivalent with respect to their skyline attributes. A tuple is skyline (under min criteria) if there is no other tuple that has a smaller *or equal* value on each skyline attribute *and* has, for some skyline attribute, a strictly smaller value. Therefore, there may be duplicates among the skyline. Such data "density" of course is characteristic of much real data. Denseness does affect skyline, but in a way

oarieli@mta.ac.il

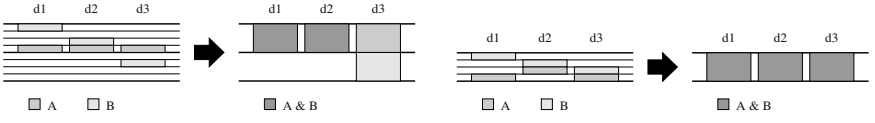
that is counter-intuitive to most. The primary effect is to *reduce* the number of skyline with respect to $\hat{\mathsf{s}}_{d,n}$, the estimated value under the sparseness condition.

So far, we have been unconcerned about the distributions of the tuples' values over each skyline column. We observe that, under sparseness, the distributions are immaterial![8] This is a remarkable characteristic of the skyline. However, under denseness, the distributions do have an effect. We consider Zipfian distributions on the columns and explain what happens. Again, the primary effect is either to reduce the number of skyline with respect to $\hat{\mathsf{s}}_{d,n}$, or not to affect it virtually at all.

Last, we consider the effects of correlation and anti-correlation among the columns. We know *a priori* that correlation must be well behaved with respect to skyline cardinality, but that anti-correlation can move the skyline count to n (with *all* the tuples being in the skyline). Yet we find that skyline cardinality is fairly stable with $\hat{\mathsf{s}} \ll n$, even in the presence of reasonably high anti-correlation.

4.1 Sparseness versus Denseness

Many attribute domains are small. For instance, a *Boolean* type only allows the values *true* and *false*. Furthermore, we are really interested in the *range* of values that occur over an attribute, rather than the domain, per se. For a given distribution of tuples, the values may cluster on just a few of the possible values. When we relax the sparseness assumption from Definition 3.1, it allows for tuples to share values. Furthermore, it allows for duplicate tuples (at least with respect to the skyline attributes). Since most real data is like this, we are interested in how this affects the skyline cardinality.



(a) Two tuples become comparable. (b) Two tuples become identical.

Fig. 6. Binning effects.

Definition 4.1. *Let* $\mathsf{s}_{\langle p \times p \rangle, n}$ *denote the random variable that measures the number of skyline tuples from a relation of* n *tuples, with respect to a* two-*dimensional skyline query. Each of the skyline attributes range over* p *values, the tuples' values are uniformally distributed over them, and the skyline attributes are statistically independent.*

[8] This is with the assumption still of statistical independence over the attributes. Note that the attributes' distributions *can* cause statistical dependence even if the attributes remain causally independent.

oarieli@mta.ac.il

Let $\mathsf{s}_{\langle p^d\rangle,n}$ more generally denote the number of skyline tuples with respect to a d-dimensional skyline query under the same conditions. Let $\hat{\mathsf{s}}_{\langle p^d\rangle,n}$ denote the expected value.

It is straightforward to establish $\hat{\mathsf{s}}_{\langle p^d\rangle,n}$'s behavior in the limit, for d, p, and n.

Theorem 4.1. Bounds on $\mathsf{s}_{\langle p^d\rangle,n}$ and $\hat{\mathsf{s}}_{\langle p^d\rangle,n}$.

a. $1 \leq \mathsf{s}_{\langle p^d\rangle,n} \leq n$

b. $\lim_{d\to\infty} \hat{\mathsf{s}}_{\langle p^d\rangle,n} = n$

c. $\lim_{p\to\infty} \hat{\mathsf{s}}_{\langle p^d\rangle,n} = \hat{\mathsf{s}}_{d,n}$

d. $\lim_{n\to\infty} \dfrac{\hat{\mathsf{s}}_{\langle p^d\rangle,n}}{n} = \dfrac{1}{p^d}$

Proof. *There must be at least one skyline tuple, and at most, there can be n. In the limit of d, all tuples will be incomparable. In the limit of p, the probability of repeat values diminishes to zero. Once $n \gg 1/p^d$, with high probability, there will be tuples with the highest value on each dimension. Just these tuples, all duplicates with respect to skyline attributes, will be skyline.* □

We are specifically interested in how $\hat{\mathsf{s}}_{\langle p^d\rangle,n}$ relates to $\hat{\mathsf{s}}_{d,n}$. Does value repetition (denseness) increase the number of expected skyline tuples, or decrease it? Denseness is equivalent to considering a relation that is initially sparse, and the tuples' values are then *binned*—that is, *partitioned*—over each attribute into just a few bins (values). So the tuples initially share no values, but after being *binned*, they do. Of course after binning, there can be duplicate *tuples* as well. Figure 6 shows two effects that occur. In some cases, a pair of tuples that were incomparable before binning may be comparable after binning. Figure 6(a) shows this.[9] Tuples A and B were incomparable before. Assume that they both are skyline. However, after binning, A trumps B. Thus only A could be in the skyline of the binned relation.[10] By this effect, it is possible to lose skyline tuples. In other cases, *all* the skyline attribute values of the two tuples become the same. Figure 6(b) shows this. Assume that A is skyline but B is not initially. Again, two incomparable tuples have become comparable upon binning, but this time both A and B might qualify as skyline afterward. By this second effect, it is possible to gain skyline tuples, due to resulting duplicate tuples.

The probability of value sharing occurring, as in Figure 6(a), is much greater generally than that of the tuples becoming equivalent, as in Figure 6(b).[11] That is, there is a higher probability that tuples will share values on some dimensions, but not on all. This gap increases with the number of dimensions. So we generally expect the first effect to dominate the second, and for the number of skyline to go down due to binning.

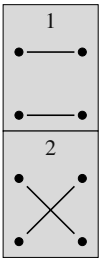
[9] The vertical axis represents the attributes' values. The horizontal axis spans the attributes. The two tuples, A and B, are shown in different shades.

[10] We say "could" because some other tuple might trump A after binning by this same argument.

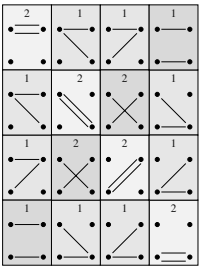
[11] This is not the case only in the extreme. For instance, if we bin all values to a single bin for each dimension, all the tuples become identical.

oarieli@mta.ac.il

There is the case when there are many more tuples, n, than possible value combinations, p^d. In this case, the second effect due to duplicates reigns. By uniformity, for each possible value combination, there is with high probability a tuple that matches it. Thus, there is a tuple with the best p value on each attribute, and so this is the only possibility for skyline. How many skyline tuples there are depends on how many duplicates of this best tuple there are. The limit in Theorem 4.1(d) reflects this effect.



(a) Edges with no repeats, $\hat{\mathsf{s}}_{2,2} = 3/2$.



(b) Edges with repeats, $\hat{\mathsf{s}}_{2\times 2,2} = 11/8$.

Fig. 7. Choose two edges.

Consider the case of two tuples of two dimensions, with each dimension as Boolean. This is the same as considering two edges placed on a 2×2 bipartite grid, as in Figure 7. If no vertex (value) sharing is allowed, the only possibilities are as in Figure 7(a). With vertex sharing, there are sixteen possibilities as in Figure 7(b) (choosing two possible edges, with replacement). By our assumptions of uniform distributions and independence, all sixteen possibilities are equally likely.

The dark-hued diagonal in Figure 7(b) represents the cases in which there are no repeated values. These behave exactly as $\mathsf{s}_{2,2}$. The light-hued diagonal are the duplicate cases, so $\hat{\mathsf{s}} = 2$ over these. The rest are cases of repeats, but no duplicates. For these, $\hat{\mathsf{s}} = 1$. Note there are twice as many cases of repeats (but no duplicates) than of duplicates.

We can solve via the probabilities for $\hat{\mathsf{s}}_{\langle p\times p\rangle,2}$ (and for higher values of n, although it becomes increasingly cumbersome).

Lemma 4.1. $\hat{\mathsf{s}}_{\langle p\times p\rangle,2} = \mathsf{H}_2 - \dfrac{1}{p} + \dfrac{3}{2p^2}$

Proof. *Follows from a case-by-case sum of the probabilities.* □

For $p > 1$, $\hat{\mathsf{s}}_{\langle p\times p\rangle,2} < \hat{\mathsf{s}}_{2,2}$.

We can establish more generally that

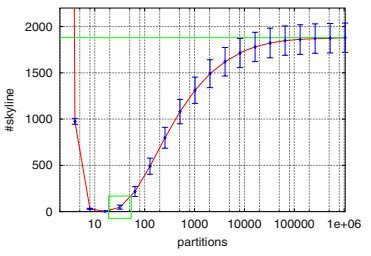
Lemma 4.2. $\hat{\mathsf{s}}_{\langle p^d\rangle,2} < \hat{\mathsf{s}}_{d,2}$, *for* $p \geq 2$ *and* $d > 1$.

oarieli@mta.ac.il

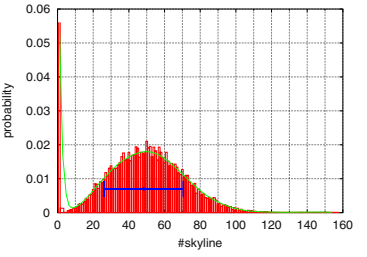
Proof. *We show* $\hat{\mathsf{s}}_{\langle p^d\rangle,2} \leq 2-1/2^{d-2}+(1/p^d)(1/2^{d-2})$ *and that* $\hat{\mathsf{s}}_{d,2} = 2-1/2^{d-1}$. *Thus,* $\hat{\mathsf{s}}_{\langle p^d\rangle,2} < \hat{\mathsf{s}}_{d,2}$ *when* $p \geq 2$ *and* $d > 1$. □

We conjecture that this is true more generally for any n, up to a point before n saturates the number of admissible values.

Conjecture 4.1. For $n < p^d \cdot \hat{\mathsf{s}}_{d,n}$, $\hat{\mathsf{s}}_{\langle p^d\rangle,n} < \hat{\mathsf{s}}_{d,n}$.



(a) Plotting "$\hat{\mathsf{s}}_{\langle p^5\rangle,10^6}$" for $d = 5$.



(b) Distribution of "$\mathsf{s}_{\langle 32^5\rangle,10^6}$".

Fig. 8. $\hat{\mathsf{s}}_{\langle p^5\rangle,10^6}$

This conjecture may be hard to prove, especially for cases of small n's and p's. For instance, while $\hat{\mathsf{s}}_{\langle p\times p\rangle,2} < \hat{\mathsf{s}}_{2,2}$ (for $p > 1$), $\hat{\mathsf{s}}_{\langle p\times p\rangle,2}$ is not monotone with respect to p. Of course, for our purposes, we are only interested in relatively large n. To ascertain experimentally $\hat{\mathsf{s}}_{\langle p^d\rangle,n}$ and the distribution of $\mathsf{s}_{\langle p^d\rangle,n}$, as in Section 3, we ran 10,000 trials for each of $p = 2$ to 2^{20} (by doubling) for $d = 5$ and $n = 10^6$. Figure 8(a) shows this. The error-bars represent the standard deviations. Figure 8(b) shows the distribution of $\mathsf{s}_{\langle 32^5\rangle,10^6}$, and is constructed in the same way as those in Figures 5(a), (b), and (c).

To the left of the nadir in Figure 8(a), the number of tuples dominates the possible value combinations, and the distribution counts the number of duplicates of that best scoring combination. These distributions are true Gaussian, by the Central Limit Theorem. (For these, we note that $\mu = \sigma^2$.) Around the nadir is the balance point between the duplicate effect and value sharing. Here, the distributions are odder, as in Figure 8(b). That distribution is bimodal, in which many trials hit the "best" value combination once (and so have a skyline of one) and many do not. As we move to the right, the distributions become well behaved and Gaussian-like again. The number of skyline is diminished due to the value-sharing effect, which acts as dimensional reduction. The distributions converge on that of $\mathsf{s}_{5,10^6}$ (1,880)—shown as the ceiling in Figure 8(a)—as p grows and the relation becomes virtually sparse.

More generally, the distribution of $\mathsf{s}_{\langle p^d,n\rangle}$ converges on that of $\mathsf{s}_{d,n}$ in the limit as p grows, and $\hat{\mathsf{s}}_{\langle p^d,n\rangle}$ converges asymptotically from below to $\hat{\mathsf{s}}_{d,n}$. Thus, $\hat{\mathsf{s}}_{d,n}$ is a ceiling on $\hat{\mathsf{s}}_{\langle p^d,n\rangle}$.

oarieli@mta.ac.il

The assumption for $\hat{\mathsf{s}}_{\langle p^d,n\rangle}$ is that any of the p bins are equally likely to have a tuple mapped to it. Thus, it is immaterial what the distribution of the attribute is, as long as our summary—the partitioning we have chosen—is an equi-width histogram of the data. This is a common way to summarize data in databases and data warehouses. Under these conditions, $\hat{\mathsf{s}}_{\langle p^d,n\rangle}$ is a good estimation.

4.2 Domain Distributions

In the basic model, we made no assumption about attributes' distributions, beyond the assumption of sparseness. Under the assumptions of sparseness and independence, the distributions are immaterial! This follows directly from Theorem 3.1 and its proof, and was illustrated in our discussion in Section 3. For one tuple to dominate another along a given dimension, the actual values do not matter, per se, just the tuples' *ranks* along that dimension.

Of course when the sparseness condition does not hold—and tuples tie on values—the skyline may no longer be independent of the attributes' distributions. In the previous section, we analyzed what happens when the sparseness assumption is relaxed, but we kept an assumption of uniform distributions. Now let us relax both the sparseness and the uniform distribution assumptions. Specifically, let us consider the domains' values under a Zipfian distribution, as this is a common distribution in real data.

Definition 4.2. *Under a* Zipfian distribution *over p values $(1,\ldots,p)$ with* zeta constant *z, the probability of value i is $1/ci^z$ (where $c = \sum_{i=1}^{p} 1/i^z$, to normalize the probabilities). Zipfian distributions are also called zeta distributions. Usually z is close to unity.*

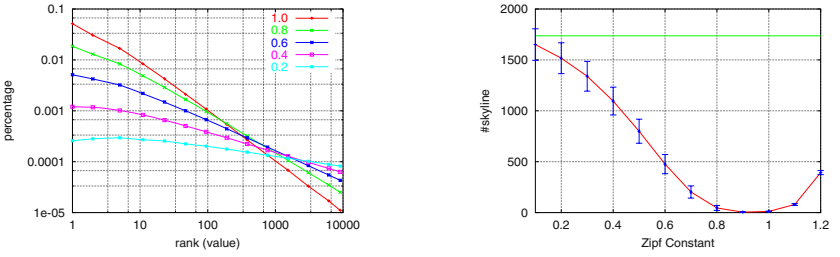
Let $\mathsf{s}_{\langle p_z^d,n\rangle}$ denote the number of skyline with respect to d min *criteria over n tuples for which the the values of each skyline attribute are distributed over p values $(1,\ldots,p)$ with a Zipfian distribution with zeta constant z, and the attributes are statistically independent.*

We ran Monte Carlo simulations as follows. For each *trial*, we generated one million tuples of d attributes randomly, each distributed with a Zipfian distribution across the integer range $1,\ldots,10,000$. A simulation then consisted of $10,000$ trials. To generate data following a Zipfian distribution, we employed the algorithm in [31]. Zipfian data when plotted on a log-log scale is linear. Figure 9(a) plots the generator's results for a million values at zeta values 1.0, .8, .6, .4, and .2, respectively, and demonstrates the fidelity of the generator, with the artifact of dropping off slightly near unity.[12]

The Zipfian distribution affects the likelihood of a tuple having a good score (near 1) on the dimension. The probability that the best possible tuple (all 1's) appears in the relation is increased, compared with the uniform distribution.

[12] We exponentially bin points together—1's, 2–3's, 4–7's, 8–15's, ...—to accommodate the log scale along the x-axis. For $z = 1.0$ we used $z = 1.001$ to avoid a singularity for $z = 1$ in the generator's algorithm.

oarieli@mta.ac.il



(a) Column's values at different z.

(b) Plotting "$\hat{\mathsf{s}}_{\langle(10^4)_z^5\rangle,10^6}$".

Fig. 9. Skyline under Zipfian distributions.

In Figure 9(b), we plot simulations for zeta values from .1 to 1.2 in .1 steps. The results are quite reminiscent to those in Figure 8(a) for varying the number of values (p). To the right of the nadir are the cases when duplicates of the best tuple constitute the skyline. As before, these are Gaussian by the Central Limit Theorem, and $\mu = \sigma^2$. Consider the simulation with $z = .8$, near the nadir in Figure 9(b). As seen in Figure 9(a), the probability of value 1 when $z = .8$ is about .037. In a million draws (generated tuples), the probability of tuple $\langle 1, 1, 1, 1, 1\rangle$ occurring at least once is 7%. The distribution of $\mathsf{s}_{\langle(10^4)_8^5,10^6\rangle}$ is bimodal and is quite similar to that for $\mathsf{s}_{\langle 32^5,10^6\rangle}$ in Figure 8(b). To the left of the nadir, the distributions become Gaussian-like again and well-behaved.

The ceiling, indicated in Figure 9(b), on which $\hat{\mathsf{s}}_{\langle(10^4)_z^5,10^6\rangle}$ is asymptotically converging is $\hat{\mathsf{s}}_{\langle(10^4)^5,10^6\rangle}$ (1,737), the expected value for 10,000 partitions but with a uniform distribution. This ceiling is lower than $\hat{\mathsf{s}}_{5,10^6}$ (1,880). The reason is that as z decreases, the Zipfian distribution resembles more a uniform one.

When we consider skyline with respect to max criteria and Zipfian distributions, we see something different. In this case for $z = .5$ (and $d = 5$ and $n = 10^6$), our simulation yields 1,791, which is above the ceiling of 1,737. However, it is still below $\hat{\mathsf{s}}_{5,10^6}$. What happens now is that the higher values are rarer, and therefore are sparser. Most skyline tuples draw on these values (since max is the criterion), so it is as if the data is sparse. This will converge asymptotically from below on $\hat{\mathsf{s}}_{5,10^6}$ as z *increases*.

More generally, $\hat{\mathsf{s}}_{\langle p_z^d,n\rangle}$ converges asymptotically from below on $\hat{\mathsf{s}}_{\langle p^d,n\rangle}$ for min criteria, and on $\hat{\mathsf{s}}_{d,n}$ for max criteria. Our arguments extend for other distributions as well. Excluding the case of duplicate saturation, $\hat{\mathsf{s}}_{d,n}$ is a ceiling on the expected skyline cardinality.

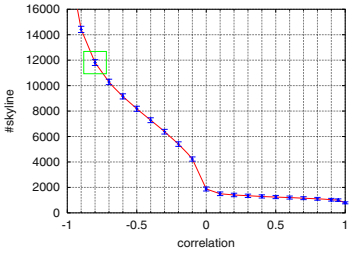
4.3 Correlation and Anti-correlation

Our other assumption in the basic model in Definition 3.1 is that of statistical independence of the dimensions (skyline attributes). This is rarely true for real data, and skyline cardinality is sensitive to correlation.

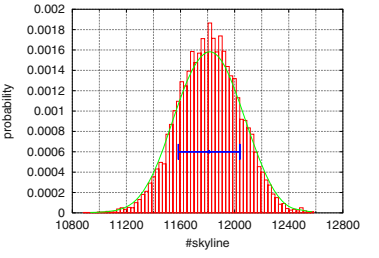
oarieli@mta.ac.il

Definition 4.3. *Let* $\mathsf{s}_{d,n}^{(r)}$ *denote the random variable that measures the number of skyline tuples of an* n*-tuple relation with respect to a* d*-dimensional skyline query, for which the* d *skyline attributes are statistically independent, save for one pair that are pair-wise correlated with correlation* r*.*

The skyline is affected quite differently by correlation and anti-correlation. High correlation between two skyline attributes acts as dimensional reduction. If tuple A has a better value on one of the dimensions than B, with high probability, it also does on the other dimension. At $r = 1.0$, that probability is one, so one of the dimensions is effectively eliminated. Anti-correlation is the antithesis of skyline, however. If A is better than B on one dimension, it is likely that B is better than A on the other. At $r = -1.0$, *all* tuples are in the skyline. In a way, skyline queries with highly anti-correlated skyline attributes do not make much sense. One is trying to optimize two values that are strictly trade-offs. Nevertheless, any realistic implementation of the skyline operator would have to accommodate such cases.



(a) Plotting "$\hat{\mathsf{s}}_{5,10^6}^{(r)}$".



(b) Distribution of $\mathsf{s}_{5,10^6}^{(-.8)}$.

Fig. 10. Skyline at different correlations.

To ascertain experimentally $\hat{\mathsf{s}}_{d,n}^{(r)}$ and the distribution of $\mathsf{s}_{d,n}^{(r)}$, as in Section 3, we ran 10,000 trials for each of $r = -0.9$ to 1.0 (and -0.95) by steps of 0.1 for $d = 5$ and $n = 10^6$. Figure 10(a) shows this. The error-bars represent the standard deviations. Figure 10(b) shows the distribution of $\mathsf{s}_{5,10^6}^{(-.8)}$, and is constructed in the same way as those in Figures 5(a), (b), and (c). The correlated cases behave as expected. Note that a simple linear interpolation between $d = 5$ and $d = 4$ would not be accurate to model the correlation. It would be easy though to fit a function for the interpolation. The $\hat{\mathsf{s}}$ do grow as anti-correlation increases, but not nearly as fast as expected. As anti-correlation increases, $\hat{\mathsf{s}}$ converges towards n very slowly. The distributions remain remarkably well behaved and Gaussian-like, and the standard deviations do not diverge rapidly.

We ought to understand further the effects of multiple correlations, and of group correlations (not just pair-wise), on skyline. To do so, it would be best

oarieli@mta.ac.il

to have a model of multiple correlations that is compatible with the correlation information as kept by relational database systems. We plan to pursue this.

5 Ramifications and Conclusions

We have found the issues of skyline cardinality to be fascinating in their own right, but our primary goal in this endeavor has been to shed light on skyline queries, their effective use, and their efficient evaluation. Our analyses and insights should help us to understand better the uses of skyline queries, and help us in this next stage to build better, more robust algorithms for skyline's computation. Understanding skyline cardinality should give us and others better insights to devising good algorithms for skyline, and for comparing average-case performances of such algorithms.

We have found that the concrete estimate $\hat{\mathsf{s}}_{d,n}$ for skyline cardinality under our basic model is a *ceiling* on the cardinality when the model's assumptions are relaxed, save with two exceptions. This means we have a cardinality estimator for use in a relational query optimizer's cost model to accommodate skyline queries. That the concrete estimate $\hat{\mathsf{s}}_{d,n}$ is a ceiling is useful; over-estimations are better than under-estimations for the query optimizer. Query performance suffers more often when the optimizer's estimation is too low.

The two exceptions to the ceiling are the extreme degenerate case when the skyline simply consists of many copies of the same best tuple and the case of extreme anti-correlation. Both these cases are somewhat antithetical to the intention of skyline queries. The optimizer can be easily made to accommodate for the first case. It can price the query as if the skyline "preferences" were regular conditions *and* do a skyline cardinality estimation, and then pick the higher estimate. More is needed for the anti-correlation case, but we note that reasonable levels of anti-correlation are not as deadly for skyline as thought.

To be sure, work remains, and many issues to resolve. Golin [6] showed that changing the shape of the *space* of the points (tuples) can dramatically affect the skyline cardinality. For us, conditions prior to the skyline operator in a complex query or integrity constraints on the database could have this effect. This requires more study, and must be understood for skyline to be fully composable with other relational operators. We need a fuller understanding of skyline under correlation, for instance how multiple and group correlations affect the skyline, and how the attributes' distributions affect the skyline's cardinality when their domains are very small (that is, p is very small).

That $\hat{\mathsf{s}}$ *diminishes* in the presence of data denseness has ramifications for users of skyline queries—and preference queries more generally—not just for the cost model. A tempting strategy when the skyline query returns too few results for the user's liking is to bin the dimensions' values further. For instance, we might see little difference between a restaurant at which the average meal is \$24 and one at which it is \$21. So we might not want one restaurant trumping another on cost unless it were at least \$5 dollars less expensive. This could lead us to bin price into five dollar brackets. However, this reasonable-seeming strategy

oarieli@mta.ac.il

backfires. As we have seen in Section 4.1, binning would only reduce further the number of choices (skyline answers).

Skyline queries offer a natural extension to min and max aggregation, and allow for one to query for nearest matches to one's objectives. The skyline operator, and potential extensions, may offer support for richer classes of queries with preferences. It may soon be worthwhile to add the skyline clause—and the underlying skyline operator—to relational systems.

We hope this work also yields more general insights into the nature of multidimensional data. In the long term, skyline may provide us, in turn, tools for the relational model and systems. For example, a skyline operator might provide the query optimizer with more efficient plans for evaluating queries with self-joins. Skyline statistics may provide a new type of useful database statistics that could yield better cost estimations for queries generally.

References

1. : Zagat Toronto Restaurants. Zagat Survey, LLC (2002)
2. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th ICDE. (2001) 421–430
3. Steuer, R.E.: Multiple Criteria Optimization: Theory, Computation, and Application. John Wiley & Sons, New York (1986)
4. Barndorff-Nielsen, O., Sobel, M.: On the distribution of the number of admissible points in a vector random sample. Theory of Probability and its Applications **11** (1966) 249–269
5. Bai, Z.D., Chao, C.C., Hwang, H.K., Liang, W.Q.: On the variance of the number of maxima in random vectors and its applications. Annals of Applied Probability **8** (1998) 886–895
6. Golin, M.J.: Maxima in convex regions. In: Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA), ACM/SIAM (1993) 352–360
7. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. JACM **22** (1975) 469–476
8. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag (1985)
9. Bentley, J.L., Kung, H.T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. JACM **25** (1978) 536–543
10. Buchta, C.: On the average number of maxima in a set of vectors. Information Processing Letters **33** (1989) 63–65
11. Bentley, J.L., Clarkson, K.L., Levine, D.B.: Fast linear expected-time algorithms for computing maxima and convex hulls. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM/SIAM (1990) 179–187
12. Matoušek, J.: Computing dominances in E^n. Information Processing Letters **38** (1991) 277–278
13. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of the 19th International Conference on Data Engineering (ICDE). (2003)
14. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: Proceedings of the 28*th* Conference on Very Large Databases (VLDB). (2002)

oarieli@mta.ac.il

15. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM Press (2003) To appear.
16. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proc. of 27th VLDB. (2001) 301–310
17. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In Carey, M.J., Schneider, D.A., eds.: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, ACM Press (1995) 71–79
18. Berchtold, S., Böhm, C., Keim, D.A., Kriegel, H.P.: A cost model for nearest neighbor search in high-dimensional data space. In: Proceedings of the Sixteenth PODS. (1997) 78–86
19. Katayama, N., Satoh, S.: The SR-tree: An index structure for high-dimensional nearest neighbor queries. In: Proceedings of Sigmod. (1997) 369–380
20. Chomicki, J.: Querying with intrinsic preferences. In: Proceedings of EDBT. (2002)
21. Chu, W.W., Yang, H., Chiang, K., Minock, M., Chow, G., Larson, C.: CoBase: A scalable and extensible cooperative information system. Journal of Intelligent Information Systems (JIIS) **6** (1996) 223–259
22. Gaasterland, T., Godfrey, P., Minker, J.: Relaxation as a platform for cooperative answering. Journal of Intelligent Information Systems (JIIS) **1** (1992) 293–321
23. Gaasterland, T., Lobo, J.: Qualifying answers according to user needs and preferences. Fundamenta Informaticæ **32** (1997) 121–137
24. Minker, J.: An overview of cooperative answering in databases. In Andreasen, T., Christiansen, H., Larsen, H.L., eds.: Third International Conference on Flexible Query Answering Systems (FQAS'98). (1998) 282–285
25. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: Proc. of Sigmod. (2000) 297–306
26. Hristidis, V., Koudas, N., Papakonstantinou, Y.: Prefer: A system for the efficient execution of multi-parametric ranked queries. In: Proceedings of Sigmod. (2001) 259–270
27. Kießling, W.: Foundations of preferences in database systems. In: Proceedings of the 28*th* Conference on Very Large Databases (VLDB). (2002)
28. Kießling, W., Köstler, G.: Preference SQL: Design, implementation, experiences. In: Proceedings of the 28*th* Conference on Very Large Databases (VLDB). (2002)
29. Roman, S.: The logarithmic binomial formula. American Mathematics Monthly **99** (1992) 641–648
30. Knuth, D.E.: Fundamental Algorithms: The Art of Computer Programming. second edn. Volume 1. Addison Wesley, Reading, MA (1973)
31. Gray, J., Sundaresan, P., Englert, S., Baclawski, K., Weinberger, P.J.: Quickly generating billion-record synthetic databases. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, ACM Press (1994)

oarieli@mta.ac.il

Query Answering and Containment for Regular Path Queries under Distortions

Gösta Grahne[1] and Alex Thomo[2]

[1] Concordia University, Montreal, Canada, grahne@cs.concordia.ca
[2] Suffolk University, Boston, USA, thomo@mcs.suffolk.edu

Abstract. We give a general framework for approximate query processing in semistructured databases. We focus on regular path queries, which are the integral part of most of the query languages for semistructured databases. To enable approximations, we allow the regular path queries to be *distorted*. The distortions are expressed in the system by using weighted regular expressions, which correspond to weighted regular transducers. After defining the notion of weighted approximate answers we show how to compute them in order of their proximity to the query. In the new approximate setting, query containment has to be redefined in order to take into account the quantitative proximity information in the query answers. For this, we define approximate containment, and its variants *k-containment* and *reliable containment*. Then, we give an optimal algorithm for deciding the k-containment. Regarding the reliable approximate containment, we show that it is polynomial time equivalent to the notorious limitedness problem in distance automata.

1 Introduction

The semi-structured data model [ABS99] is now widely used as a foundation for reasoning about a multitude of applications, where the data is best formalized in terms of labeled graphs. Such data is usually found in Web information systems, XML data repositories, digital libraries, communication networks, and so on.

Regarding the query languages for semi-structured data, virtually all of them provide the possibility for the user to query the database through regular expressions. The design of query languages using regular expressions is based on the observation that many of the recursive queries, which arise in practice, amount to graph traversals. In essence, these queries are graph patterns, and the answers to the query are subgraphs of the database that match the given pattern [MW95,ABS99,C+99,C+00]. In particular, the (sub)queries expressed by regular expressions are called *regular path queries*.

For example, for answering the query

$$Q = \_^* \cdot article \cdot \_^* \cdot ref \cdot \_^* \cdot article.hopcroft$$

one should find all the paths having at some point an edge labeled *article*, followed by any number of other edges then by an edge *ref* followed at some point

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 98–115, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

by an edge *article* and immediately after concluding with an edge labeled with *hopcroft*.

However, we are often willing to live with structural information that is approximate. In other words, the semistructured data represented by a graph database can be an approximation of the real world, rather than an exact representation. On the other hand, the user herself can have an approximate idea and/or knowledge about the world, and this has as a consequence a need for non exact information to be extracted from the database. In both cases the conclusion is that we need to deal with approximate queries and databases, and give approximate answers to the user queries. As an example, suppose that the user gives the above query, but in the database we have edges labeled *papers* instead of *article* or we have recorded in the database only books by Hopcroft, and no papers authored by him. In both cases, the user would get an empty answer under exact query semantics, while it would be very desirable if the system had the ability to substitute *article* by *paper* "for free," and to substitute *article* by *book* with some "cost," say 3. The system could then warn the user about the distortions, by producing a query answer, weighted by the distortion cost. The database system administrator could capture such allowed distortions by building a weighted regular expression

$$(\Delta, 0, \Delta)^*.((article, 1, paper) + (article, 3, book)).(\Delta, 0, \Delta)^*$$

This regular expression is defined over symbol-weight-symbol triplets (R, k, S), where k is the semantic "cost" of distorting R to S[1], and $(\Delta, 0, \Delta)$ is a shorthand for $\sum_{R \in \Delta}(R, 0, R)$, with Δ being the underlying (finite) alphabet. It is easy to see that such extended regular expressions exactly correspond to *weighted transducers*, if we think of the transducers as finite automata over symbol-weight-symbol triplets.

For simplicity, we can also allow the system administrator to use word-weight-word triplets (v, k, w) for building such weighted regular expressions. It can easily be shown that the (corresponding) weighted transducers over such word-weight-word can be transformed into transducers over symbol-weight-symbol triplets. Although the weighted regular expressions over word-weight-word triplets are equivalent to the ones over symbol-weight-symbol triplets, the former are easier to use when we want to capture path structural distortion, as for example

$$(automata.book.author.hopcroft, 1, automata.book.author.ullman).$$

We can even allow full general regular expressions in the above triplets as for example

$$(automata._^*.book._^*.hopcroft, 1, automata._^*.book._^*.ullman).$$

The semantics of such triplets is that we can distort any word in the language of the first regular expression to any database path spelling a word in the language

[1] R or S could be ϵ as well, but not both.

oarieli@mta.ac.il

of the second. It can be shown, that we are still able to find an equivalent weighted regular expression over the symbol-weight-symbol triplets.

In this paper, we formally define the notion of weighted approximate answers to regular path queries. Given such a query and having available a weighted transducer (through a weighted regular expression) we show that we can effectively compute all the approximate answers on a database. Furthermore, we can produce the approximate answers in increasing order of their weight, i.e. from the less to the more distorted.

The similar problem of finding approximate patterns in sequence databases is treated in depth in [JMM95]. There, Jagadish, Mendelzon and Milo formalized a very powerful rule-based system through which one can specify the possible allowed distortions of a word to some other word. Unfortunately, their distortion model has an undecidable word problem. Hence, should we use the model of [JMM95], we would not be able to decide in general the membership of a tuple in the approximate answer to a query.

We can say that, the motivation for using weighted regular expressions (i.e. weighted transducers) as a distortion model is similar to the motivation for using regular expressions for querying recursive graph patterns and not the more powerful formalisms such as context-free rule-based grammars.

Having built our query approximation framework, we turn to defining a query containment notion, which takes into account the quantitative distortion information available in the tuples of the answer sets. For this, we define the *approximate containment*, and its variants k-containment, and *reliable* containment. For the first notion, we say that a query is approximately contained in a another query, if for *any* database the tuples for the first query are also tuples for the second one, and furthermore, in the second query, those tuples are more reliable, i.e. they are obtained through less query distortion. The reason behind this view is that, since for obtaining a tuple, the first query needs more distortion than the second one, semantically the first is "smaller" than the second. However, as we show, this unrestricted notion of approximate query containment does not help to much. Hence, in addition, we shall require that on *any* database the (distortion) weight of the tuples for the first query to not be greater than a given number, say k, compared to the weight of the corresponding tuples for the second query. We call this k-containment, and we give an optimal algorithm, based on algebraic properties of automata, for deciding such containment between two given queries.

Depending on the application, we might be interested only in the *existence* of the above number k. Namely, we would like to know, for two given queries and a distortion transducer, whether there exists a number k, such that on any database, the (distortion) weight of the tuples for the first query is not greater than k compared to the weight of the corresponding tuples for the second query. We call this variant reliable containment, and show that it is polynomial time equivalent with the intricate limitedness problem for distance automata, intensely investigated by Hashiguchi and others [Has82,Has90,Has00, Leu91,Sim94].

oarieli@mta.ac.il

2 Basic Definitions

We consider a database to be an edge labeled graph. This graph model is typical in semistructured data, where the nodes of the database graph represent the objects and the edges represent the attributes of the objects, or relationships between the objects.

Formally, let Δ be a finite alphabet. Elements of Δ will be denoted $R, S, \ldots$. As usual, Δ^* denotes the set of all finite words over Δ. Words will be denoted by $u, w, \ldots$. We also assume that we have a universe of objects, and objects will be denoted $a, b, c, \ldots$. A *database* DB is then a graph (V, E), where V is a finite set of objects and $E \subseteq V \times \Delta \times V$ is a set of directed edges labeled with symbols from Δ. Figure 1 shows an example of a database. If there is a path labeled $R_1, R_2, \ldots, R_k$ from a node a to a node b we write $a \xrightarrow{R_1 R_2 \ldots R_k} b$.

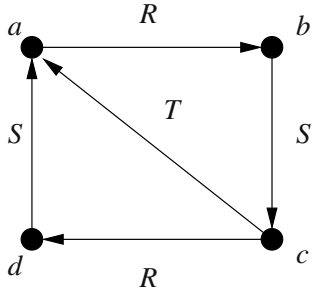


Fig. 1. An example of a graph database

A *query* Q is a regular language over Δ. Let Q be a query and $DB = (V, E)$ a database. Then, the *exact answer* to Q on DB is defined as

$$ans(Q, DB) = \{(a, b) : \{a, b\} \subseteq V \times V \text{ and } a \xrightarrow{w} b \text{ in } DB \text{ for some } w \in Q\}.$$

For instance, if DB is the graph in Figure 1, and $Q = \{SR, T\}$, then $ans(Q, DB) = \{(b, d), (d, b), (c, a)\}$.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. A *weighted transducer* $\mathcal{T} = (P, \Delta, \tau, P_0, F)$ consists of a finite set of states P, an input/output alphabet Δ, a set of starting states P_0, a set of final states F, and a transition relation $\tau \subseteq P \times \Delta^* \times \Delta^* \times \mathbb{N} \times P$.

An example of a weighted transducer $(\{p_0, p_1, p_2\}, \{R, S, T\}, \tau, \{p_0\}, \{p_2\})$ is shown in Figure 2. Intuitively, for instance $(p_0, RT, RS, 2, p_1) \in \tau$ means that if the transducer is in state p_0 and reads word RT, it emits the word RS at cost 2 and goes to state p_1.

Given a weighted transducer $\mathcal{T} = (P, \Delta, \tau, P_0, F)$, and a word $u \in \Delta^*$ we say that a word $w \in \Delta^*$ is an *output of* $\mathcal{T}$ *for* u *through a* k*-weighted distortion* if there exists a sequence $(p_0, u_1, w_1, k_1, p_1)$, $(p_1, u_2, w_2, k_2, p_2)$, $\ldots$, $(p_{n-1}, u_n, w_n, k_n, p_n)$ of state transitions of τ, such that $p_1 \in P_0$, $p_n \in F$,

oarieli@mta.ac.il

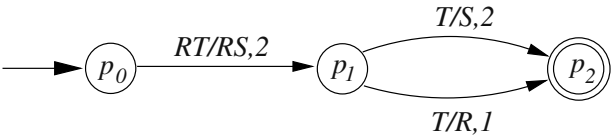


Fig. 2. A weighted transducer.

$u = u_1 \dots u_n$, $w = w_1 \dots w_n$, and $k = k_1 + \cdots + k_n$. We denote the set of all outputs of $\mathcal{T}$ for u (regardless of distortion) by $\mathcal{T}(u)$. For a language $L \subseteq I^*$, we define $\mathcal{T}(L) = \bigcup_{u \in L} \mathcal{T}(u)$. Later we will also use the notation $rel(\mathcal{T})$ to denote the set of all pairs $(u, w) \in \Delta^* \times \Delta^*$, where w is an output of $\mathcal{T}$ when providing u as input. Similarly, $dom(\mathcal{T})$ and $ran(\mathcal{T})$, will be used to denote the domain and range of $rel(\mathcal{T})$.

Given a weighted transducer $\mathcal{T}$, and words u and w, the $\mathcal{T}$-*distance* between u and w is defined as

$$d_{\mathcal{T}}(u,w) = \begin{cases} inf\{k : w \text{ is an output of } \mathcal{T} \text{ for } u \text{ through a } k\text{-weighted distortion}\} \\ \infty, \text{ if } w \notin \mathcal{T}(u). \end{cases}$$

Now, the *approximate answer* of Q on DB, through a distortion transducer $\mathcal{T}$, is defined as

$$ans_{\mathcal{T}}(Q, DB) = \{(a, b, k) \in V \times V \times \mathbb{N} : \\ k = inf\{d_{\mathcal{T}}(u, w) : u \in Q \text{ and } a \xrightarrow{w} b \text{ in } DB\}\}$$

For example, in the database DB of Figure 1, if $Q = \{RTT\}$, and the distortion transducer is as in Figure 2, then $\mathcal{T}(Q) = \{RSR, RSS\}$. We thus have $ans(Q, DB) = \emptyset$, while $ans_{\mathcal{T}}(Q, DB) = \{(a, d, 3), (c, b, 3)\}$.

For a query Q we want to get also the query itself from the transduction. For this reason we will usually consider that the distortion transducers also have the ability to "leave everything unchanged." This can be easily achieved automatically by the system, which can add to a distortion transducer a new state, say p_{id}, that is both initial and final, as well as the neutral transitions $\{(p_{id}, R, R, 0, p_{id}) : R \in \Delta\}$. Also, for technical reasons we will require that there are no "free-distortion" transitions of the form $(p, R, S, 0, q)$, $(p, R, \epsilon, 0, q)$, or $(p, \epsilon, R, 0, q)$, where $R \neq S$. Clearly, this restriction although done for technical reasons, closely reflects the reality where we should warn the user that we have indeed somehow distorted her original query in order to obtain the produced tuple(s). Notably, all our lower complexity bounds will be derived by considering this class of distortion transducers.

A transducer $(P, \Delta, \tau, P_0, F)$ is said to be in the *standard form* if τ is a relation over $P \times (\Delta \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{N} \times P$. Intuitively, the standard form restricts the input and output of each transition to be only a single letter or ϵ. We call such transitions *elementary* transitions. It is easy to see that any weighted regular transducer is equivalent to a weighted regular transducer in standard form. The transducer transformation is done by applying the following two steps.

oarieli@mta.ac.il

In the first step, we eliminate the transitions of the form $(p, w, R_1 \dots R_n, h, q)$. For this, we introduce new states $p_1, \dots, p_{n-1}$ and replace such a transition by the elementary transitions $(p, w, R_1, 1, p_1)$, $(p_1, \epsilon, R_2, 1, p_2)$, …, $(p_{n-1}, \epsilon, R_n, n - h + 1, q)$. It is easy to see that we can assume without loss of generality that $n > h - 1$. If not we can multiply all the transition weights by an integer factor and have the weight greater than corresponding word length. At the end, we divide by these factors[2] the weight of the produced tuples.

In the second step, we eliminate transitions of the form $(p, S_1 \dots S_m, R, k, q)$. For this, we introduce new states $p_1, \dots, p_{m-1}$ and replace such a transition by the elementary transitions $(p, S_1, R, 1, p_1)$, $(p_1, S_2, \epsilon, 1, p_2)$, …, $(p_{m-1}, S_m, \epsilon, m - k + 1, q)$. Similarly, we assume without loss of generality that $n > k - 1$.

3 Computing Approximate Answers

A graph database can be seen as an NFA where the graph nodes are the automaton states and all states are both initial and final. Seen from another perspective, in the "classical" case of exact semantics (see [MW95,ABS99]), computing $ans(Q, DB)$ given the automata $\mathcal{A}_Q$ for Q and $\mathcal{A}_{DB}$ for the database, essentially amounts to constructing the Cartesian product automaton $\mathcal{A}_Q \times \mathcal{A}_{DB}$ (in a lazy way) and outputting the pair (a, b), if and only if there is, in the Cartesian product automaton, an initial state $(\_, a)$ leading to a final state $(\_, b)$.

We show next that for computing $ans_{\mathcal{T}}(Q, DB)$ we can construct an automaton from the Cartesian product of $\mathcal{A}_Q$, $\mathcal{T}$, and $\mathcal{A}_{DB}$. The approximate answer can then be read from this automaton, similarly to the "classical" case.

Let $\mathcal{A}_Q = (P_Q, \Delta, \tau_Q, P_{0_Q}, F_Q)$ be an ϵ-free NFA that accepts Q, and let $\mathcal{T} = (P_{\mathcal{T}}, \Delta, \tau_{\mathcal{T}}, P_{0_{\mathcal{T}}}, F_{\mathcal{T}})$ be the distortion transducer in standard form. Considering the database DB as another ϵ-free NFA, $\mathcal{A}_{DB} = (P_{DB}, \Delta, \tau_{DB}, P_{DB}, P_{DB})$, we construct the transducer $\mathcal{C} = (P, \Delta, \tau, P_0, F)$, where $P = P_Q \times P_{\mathcal{T}} \times P_{DB}$, $P_0 = P_{0_Q} \times P_{0_{\mathcal{T}}} \times P_{DB}$, $F = F_Q \times F_{\mathcal{T}} \times P_{DB}$, and the transition relation τ is defined by, for $(p, q, r) \in P$ and $R, S \in \Delta$,

$$
\begin{aligned}
\tau = \{&((p,q,r), R, S, k, (p',q',r')) : \\
&(p, R, p') \in \tau_Q, (q, R, S, k, q') \in \tau_{\mathcal{T}}, (r, S, r') \in \tau_{DB}\} \cup \\
&\{((p,q,r), \epsilon, S, k, (p,q',r')) : (q, \epsilon, S, k, q') \in \tau_{\mathcal{T}}, (r, S, r') \in \tau_{DB}, p \in P_Q\} \cup \\
&\{((p,q,r), R, \epsilon, k, (p',q',r)) : (p, R, p') \in \tau_Q, (q, R, q', \epsilon, k) \in \tau_{\mathcal{T}}, r \in P_{DB}\}.
\end{aligned}
$$

It is easy to see that $(a, b, k) \in ans_{\mathcal{T}}(Q, DB)$, if and only if there exists, in the graph representation of $\mathcal{C}$, a final state $(\_, \_, b)$ reachable from an initial state $(\_, \_, a)$, and the shortest path between them has cost k.

For shortest paths, both Dijkstra's algorithm and the Floyd-Warshall algorithm (see e.g. [AHU74]) could be used. Although the running times for both Dijkstra's and Floyd-Warshall algorithms are asymptotically the same, perhaps Dijkstra's algorithm is better suited in our scenario. The first reason is that

[2] We might have done several such multiplications.

oarieli@mta.ac.il

in practice the user might be interested in computing objects reachable only from a limited number of objects, for example when we have a rooted database graph. In such a case, the running time of Dijkstra's algorithm is better, since we don't need to compute the shortest paths between all pairs of objects, as in the Floyd-Warshall algorithm.

The second reason is that most of the time the user is interested only in receiving, say, the 20 best answers. Then, the Dijkstra's algorithm is the ideal choice: It processes the nodes in the order of their distance from the source. Hence, if we could construct the transducer $\mathcal{C}$ on the fly, while at the same time applying the Dijkstra algorithm, then we could stop the execution of the algorithm after 20 iterations.

We can construct the transducer $\mathcal{C}$ on the fly by utilizing a lazy algorithm similar to the lazy query evaluation algorithm of [ABS99], which in essence constructs the Cartesian product of a query with a database. Notably, the Dijkstra algorithm can be elegantly combined with such a lazy construction of $\mathcal{C}$. By assuming in addition a temporary cache of the "so far reached" objects (the set *reach* in the afore mentioned book), we can avoid accessing the same object in the database more than once. Because of space limitation, we omit the presentation of such a query evaluation algorithm.

4 Containment

In this section, we define and study three notions of containment for regular path queries under approximate semantics.

Recall, that a query Q_1 is (in the usual sense) *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff $ans(Q_1, DB) \subseteq ans(Q_2, DB)$, for all DB's [GT01]. It is easy to see that this notion of query containment coincides with the (algebraic) language containment of Q_1 and Q_2, i.e. $Q_1 \sqsubseteq Q_2$ iff $Q_1 \subseteq Q_2$. However, under approximate semantics the tuples in the query answers are weighted, and so the containment should take into consideration the tuple weights.

As we know, the smaller the weight of a tuple the better or more reliable it is. For out first notion of containment, we say that a query Q_1 is *approximately contained* in a query Q_2, if for any database the answer-tuples for Q_1 are also answer-tuples for Q_2, and furthermore, under Q_2, those tuples are more reliable. The reason behind this view is that, since for obtaining a tuple, Q_1 needs more distortions than Q_2, semantically Q_1 is "smaller" than Q_2.

However, the approximate query containment is perhaps not very useful. This is because the distance between the corresponding tuples can be arbitrarily large. In other words, it could happen that, for any $n \in \mathbb{N}$ we can find a database such that the tuples obtained for Q_1 on this database, have a distortion weight greater than n compared to the weight of the corresponding tuples for Q_2 obtained on the same database.

Hence, we are also interested in the *quality* of the approximate query containment. For this, we define the *k-containment*, which in addition to approximate containment requires that the weights of the corresponding tuples do not *differ*

oarieli@mta.ac.il

more than a given number k. Also, depending on the application, the mere existence of such a number k could be useful to know. For this, we define the *reliable containment*, which asks whether or not there exists a number k, for which the k-containment holds. Formally, we have

1. A query Q_1 is *approximately contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$, when for *any* database DB, if $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, then $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$ and $m \leq n$.
2. A query Q_1 is *k-contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$, if in the above we also have that $n - m \leq k$.
3. A query Q_1 is *reliably contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T},\omega} Q_2$, if there exists a $k \in \mathbb{N}$, such that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$.

Surprisingly enough, the (unbounded) approximate containment does not offer more information in reasoning about queries, than the containment under exact semantics. Namely, we show that for any distortion transducer $\mathcal{T}$

Theorem 4.1. *$Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ iff $Q_1 \subseteq Q_2$.*

Proof "If." From the definition of the approximate answers through a distortion transducer, we have that for any database DB, if $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, there exists a word $w \in Q_1$, and a word u such that $a \xrightarrow{u} b$ in DB, and $d_{\mathcal{T}}(w, u) = n$. Since $Q_1 \subseteq Q_2$, we have that $w \in Q_2$ as well, and so, for sure there exists an m not bigger than n (i.e. $m \leq n$) such that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$.

"Only if." We show that $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ implies $Q_1 \subseteq Q_2$. Let $w = R_1 \dots R_h$ be a word in Q_1. From w we construct a canonical database DB with vertices $\{a, c_1, \dots, c_{h-1}, b\}$ and edges $\{(a, R_1, c_1), \dots, (c_{h-1}, R_h, b)\}$. Clearly, $(a, b, 0) \in ans_{\mathcal{T}}(Q_1, DB)$, and from $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ we have that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$, where $m \leq 0$, i.e. $m = 0$. So, there is a word in Q_2 that without being distorted at all can label a path from a to b in DB. Since there is only one path between a and b in DB and this path spells w, we have that $w \in Q_2$. □

In the rest of the paper, we will be interested in the k- and reliable containments because of their practical usability.

Although related, the problems of k-containment and the reliable containment are different. For the k-containment problem, the input is two queries, a distortion transducer, and a fixed number k that the user provides. Then, the question is whether the queries are at most "k steps apart," or not. On the other hand, for the reliable containment problem, k is not part of the input, and the question is existential.

Now, we will define the $\mathcal{T}$-distance between two queries, and then give a necessary and sufficient condition for the k- and reliable containment, based on their $\mathcal{T}$-distance.

Consider a word w on Δ. The $\mathcal{T}$-distance between Q_1 and w is

$$d_{\mathcal{T}}(Q_1, w) = inf\{d_{\mathcal{T}}(u, w) : u \in Q_1\}.$$

oarieli@mta.ac.il

Based on that, the $\mathcal{T}$-distance between Q_1 and Q_2 can be naturally defined as

$$d_{\mathcal{T}}(Q_1, Q_2) = sup\{d_{\mathcal{T}}(Q_1, w) : w \in Q_2\}.$$

Returning to our problem, we give the following characterization.

Theorem 4.2.

1. *$Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ if and only if $Q_1 \subseteq Q_2$ and $d_{\mathcal{T}}(Q_1, Q_2) \leq k$.*
2. *$Q_1 \sqsubseteq_{\mathcal{T},\omega} Q_2$ if and only if $Q_1 \subseteq Q_2$ and there is a $k \in \mathbb{N}$, such that $d_{\mathcal{T}}(Q_1, Q_2) \leq k$.*

Proof. We will prove only the first claim, since the second one follows directly from the first.

"If." Let DB be a database and $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$. Since $Q_1 \subseteq Q_2$ we have that there exists a $m \leq n$, such that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$. Now, we want to prove that $n - m \leq k$. Since $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$, there exists a word $w_2 \in Q_2$ such that $d_{\mathcal{T}}(w_2, u) = m$, for some $a \xrightarrow{u} b$ in DB. Now, from the condition $d_{\mathcal{T}}(Q_1, Q_2) \leq k$, we have that for the word w_2, there exists a word $w_1 \in Q_1$, such that $d_{\mathcal{T}}(w_1, w_2) \leq k$. In plain language, w_1 needs less than k transducer distortions to become w_2. Hence, we finally have

$$\begin{aligned} n &\leq inf\{d_{\mathcal{T}}(w_1, u) : a \xrightarrow{u} b \text{ in } DB\} \\ &\leq k + inf\{d_{\mathcal{T}}(w_2, u) : a \xrightarrow{u} b \text{ in } DB\} \\ &= k + m, \end{aligned}$$

i.e. $n - m \leq k$.

"Only if." The fact that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ implies $Q_1 \subseteq Q_2$ follows directly from Theorem 4.1. Now, we continue showing that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ implies $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. Let $w = R_1 \dots R_l$ be a word in Q_2. We construct a canonical database DB with vertices $\{a, c_1, \dots, c_{l-1}, b\}$ and edges $\{(a, R_1, c_1), \dots, (c_{l-1}, R_l, b)\}$. Clearly, $(a, b, 0) \in ans_{\mathcal{T}}(Q_2, DB)$ and from $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ we have that $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, where $n \leq k$. Observe that, there is only one path between a and b in DB and this path spells w. So, we have that $d_{\mathcal{T}}(Q_1, w) = n$, i.e. $d_{\mathcal{T}}(Q_1, w) \leq k$. Since w was an arbitrary word in Q_2, we finally get that $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. □

Now we will focus on reasoning about $d_{\mathcal{T}}(Q_1, Q_2)$. Let $\mathcal{A}_1 = (P_1, \Delta, \tau_1, P_{0_1}, F_1)$ and $\mathcal{A}_2 = (P_2, \Delta, \tau_2, P_{0_2}, F_2)$ be two ϵ-free automata for Q_1 and Q_2 respectively, and let $\mathcal{C} = \mathcal{A}_2 \times \mathcal{T} \times \mathcal{A}_1 = (P, \Delta, \tau, P_0, F)$ be a Cartesian product transducer constructed as in Section 3.

For simplicity of exposition, we we will call a sequence of transitions a *path* (not necessarily simple) in the transducer. Suppose that a path π is the sequence of transitions $(p_1, u_1, w_1, k_1, p_2)$, $(p_2, u_2, w_2, k_2, p_3)$, $\dots$, $(p_n, u_n, w_n, k_n, p_{n+1})$. We say that π *spells* $u_1 \dots u_n$ *as input*, and denote this as $in(\pi) = u_1 \dots u_n$. Additionally, we say that π *spells* $w_1 \dots w_n$ *as output*, and denote this as $out(\pi) = w_1 \dots w_n$. Finally, we say that π has *weight* k, and denote this as $weight(\pi) = k$, if $k = k_1 + \dots + k_n$.

oarieli@mta.ac.il

The following lemma says that measuring the distortion through the transducer $\mathcal{T}$ or $\mathcal{C}$ is in essence the same.

Lemma 4.1. *For $u, w \in \Delta^*$ we have that $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w)$*

PROOF. From the construction of the Cartesian product transducer $\mathcal{C}$, if there is a path π in $\mathcal{T}$, such that $in(\pi) = u$ and $out(\pi) = w$, then π is also in $\mathcal{C}$ and vice versa. Hence, we get $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w)$. Otherwise, if there is not such a path, we can easily see that $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w) = \infty$. □

Now, consider the weighted automaton $\mathcal{A}$, that we get if we project out the input column of the transition relation of $\mathcal{C}$. Formally, $\mathcal{A} = (P, \Delta, \tau_{\mathcal{A}}, P_0, F)$, where $\tau_{\mathcal{A}} = \{(p, R, k, q) : (p, S, R, k, q) \in \tau\}$. Let p and q be two states of $\mathcal{A}$, and let π be a path between them. Suppose $out(\pi) = w$. Note that there can be more than one path[3] between p and q spelling w. In reasoning about the k-containment we will be interested in the "best" path(s) spelling w, i.e. the one(s) with the smallest weight. Let therefore

$$d_{\mathcal{A}}(p, w, q) = inf\{weight(\pi) : \pi \text{ is a path spelling } w \text{ and going from } p \text{ to } q \text{ in } \mathcal{A}\}.$$

Now, we define the *distance* of $\mathcal{A}$, as

$$d(\mathcal{A}) = sup\{d_{\mathcal{A}}(p, w, p) : w \text{ is accepted by } \mathcal{A}, p \in P_0, q \in F\}.$$

Based on the these definitions, Lemma 4.1, and the construction of the weighted automaton $\mathcal{A}$, the following theorem can be easily verified.

Theorem 4.3. $d_{\mathcal{T}}(Q_1, Q_2) = d(\mathcal{A})$.

We say that, a weighted automaton $\mathcal{A}$ is *k-limited* (for a given k) if $d(\mathcal{A}) \leq k$, and $\mathcal{A}$ is *limited* if there *exists* a $k \in \mathbb{N}$, such that $\mathcal{A}$ is k-limited.

Now, Theorem 4.3 along with Theorem 4.2, say that the k-containment (reliable containment) is reducible to the k-limitedness (limitedness) of weighted automata. Since such an automaton is constructible in polynomial time on the size of Q_1, Q_2, and $\mathcal{T}$, we have that the afore mentioned reduction is polynomial as well.

If we restrict ourselves in weighted automata without ϵ-transitions, we get a class of automata, which are widely known as *distance automata*, and whose limitedness problem is well known for its intricacy. The first solution was obtained by Hashiguchi in 1982, and it gave him the key for solving the star height problem, that had been open for over two decades. Hashiguchi's solution runs in doubly exponential time. By now, it is known that the problem is PSPACE hard [Leu91]. The best known algorithm for deciding whether a distance automaton is limited is by Leung [Leu91] and it runs in single exponential time.

Leung's algorithm is based on the notion of "distance matrices" which can elegantly capture the behavior of distance automata. However, the fact that the

[3] Such paths could have some ϵ-transitions as well.

oarieli@mta.ac.il

distance automata are ϵ-free is of essential importance in using Leung's distance matrices.

In order to make use of Leung's algorithm for deciding the reliable containment (which corresponds to the limitedness problem), we show in Section 6, that we can efficiently transform the transducer $\mathcal{C}$ into one with ϵ-free output transitions, while preserving the semantics of $\mathcal{C}$. Consequently, the automaton $\mathcal{A}$, that we obtain from this ϵ-free output transducer, will be ϵ-free as well.

Unfortunately, Leung's algorithm is unable to decide the k-limitedness of a distance automaton, which in turn is needed to decide the k-containment of queries. Also, as far as the authors know, there is no previous work on k-limitedness of distance automata.

In the next section, we provide an optimal solution to the k-limitedness problem, by automata constructs. Our solution is applicable to general weighted automata, as opposed to only distance automata.

5 Deciding k-Containment

We consider the automaton $\mathcal{A}$, constructed in the previous section, having as weights on its transitions only 0 and 1. If not, we can easily "normalize" it by replacing each transition (p, R, m, q), where $m > 1$, by the sequence of transitions $(p, R, 1, r_1)$, $(r_1, \epsilon, 1, r_2)$, ..., $(r_{m-1}, \epsilon, 1, q)$. For technical reasons, we also add to the transition relation of $\mathcal{A}$ the neutral transitions $(p, \epsilon, p, 0)$ for each state, i.e. self-loops of weight 0, and labeled with ϵ. Evidently, these neutral transitions do not alter any salient features of $\mathcal{A}$. However, we can now assume that for any two transitions in the automaton $\mathcal{A}$ there is always a 0-weighted transition between them.

We will need a few simple operations on automata. Let $\mathcal{A}$ and $\mathcal{B}$ be automata. Then we denote with $\mathcal{A} \cup \mathcal{B}$ the automaton, obtained by the usual construction, recognizing $L(\mathcal{A}) \cup L(\mathcal{B})$. Similarly, $\mathcal{A} \bullet \mathcal{B}$, denotes the automaton recognizing $L(\mathcal{A}).L(\mathcal{B})$.

Now, let's assume that all automata have their states labeled by consecutive integers starting from 1. We denote with $\mathcal{A}_{i,j}$ the automaton obtained from $\mathcal{A}$, by shifting the set of initial states to be $\{i\}$ and the final states to be $\{j\}$. Also, let $\mathbf{0}(\mathcal{A})$ be the automaton obtained from $\mathcal{A}$ by deleting all transitions with cost 1. Finally, for $\{i, j\} \subset \{1, \ldots, n\}$, we consider the set of elementary automata $\mathbf{1}_{i,j}(\mathcal{A})$, each obtained from $\mathcal{A}$ by retaining only transitions between i and j, and only those that have cost 1. Observe that, an automaton, say $(\mathbf{0}(\mathcal{A}))_{i,j}$, can be a full-fledged automaton i.e. with loops, while an elementary automaton, say $\mathbf{1}_{i,j}(\mathcal{A})$, is simple in the sense that it does not contain any loops.

Given a normalized weighted automaton $\mathcal{A} = (\{1, \ldots, n\}, \Delta, \tau, S, F)$, we wish to compute an automaton $\mathbf{k}(\mathcal{A})$, such that $L(\mathbf{k}(\mathcal{A})) = \{w \in L(\mathcal{A}) : d_{\mathcal{A}}(w) \leq k\}$.

Clearly, if we are able to construct $\mathbf{k}(\mathcal{A})$, then we can decide whether or not $d(\mathcal{A}) \leq k$, by testing the language equality $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$. Hence, by this, we cast the decision of the (weighted) k-containment into a pure regular language equivalence test, which can be done in polynomial space.

oarieli@mta.ac.il

We will construct $\mathbf{k}(\mathcal{A})$ by a recursive algorithm obtained from the following equations:

$$\mathbf{k}(\mathcal{A}) = \mathcal{A}^0 \cup \mathcal{A}^1 \cup \ldots \cup \mathcal{A}^k$$

where $\mathcal{A}^0 = \mathbf{0}(\mathcal{A})$, and for $1 \leq h \leq k$

$$\mathcal{A}^h = \bigcup_{i \in S, j \in F} \mathcal{A}^h_{i,j}$$

where

$$\mathcal{A}^h_{i,j} = \begin{cases} \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j} & \text{for } h \text{ even} \\ \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{(h-1)/2}_{i,m} \bullet \mathcal{A}^{(h+1)/2}_{m,j} & \text{for } h \text{ odd} \end{cases}$$

for $h > 1$, and

$$\mathcal{A}^1_{i,j} = \bigcup_{\{m,l\} \subset \{1,\ldots,n\}} (\mathbf{0}(\mathcal{A}))_{i,m} \bullet \mathbf{1}_{m,l}(\mathcal{A}) \bullet (\mathbf{0}(\mathcal{A}))_{l,j} .$$

We can now show that indeed:

Theorem 5.1. $L(\mathbf{k}(\mathcal{A})) = \{w \in L(\mathcal{A}) : d_{\mathcal{A}}(w) \leq k\}$.

Proof. We will prove that for all $0 \leq h \leq k$, the automaton $\mathcal{A}^h$, considered as graph, consists of *all* the paths[4] in $\mathcal{A}$ with weight exactly h, and going from an initial to a final state. So, a word spelled by such a path cannot have distance more than h. As a consequence the automaton

$$\mathbf{k}(\mathcal{A}) = \mathcal{A}^0 \cup \mathcal{A}^1 \cup \ldots \cup \mathcal{A}^k$$

will accept *all* the words in $L(\mathcal{A})$ that cannot have $\mathcal{A}$-distance more than $0, 1, \ldots, k$. From this, our claim follows.

We proceed by induction on h. For $h = 0$, the automaton $\mathcal{A}^0$ is in fact $\mathbf{0}(\mathcal{A})$, so it consists of all the 0-weighted paths in $\mathcal{A}$, going from some initial to some final state. For $h = 1$, the automaton, say

$$\mathcal{A}^1_{i,j} = \bigcup_{\{m,l\} \subset \{1,\ldots,n\}} (\mathbf{0}(\mathcal{A}))_{i,m} \bullet \mathbf{1}_{m,l}(\mathcal{A}) \bullet (\mathbf{0}(\mathcal{A}))_{l,j} ,$$

consists of the $\mathcal{A}$-paths starting from state i and traversing any number of 0-weighted arcs (transitions) up to some state m, then a 1-weighted arc going to some state l, and after that, any number of 0-weighted arcs ending up in state j. Since m and l range over *all* the possible states, we have that the above described paths are in fact *all* the 1-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^1$ will consists of all the 1-weighted paths of $\mathcal{A}$ going from an initial to a final state.

[4] By a path we do not necessarily mean a simple path. Some authors use the term walk instead.

oarieli@mta.ac.il

For $h = 2$, the automaton, say

$$\mathcal{A}^2_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^1_{i,m} \bullet \mathcal{A}^1_{m,j},$$

consists of concatenations of the 1-weighted paths of $\mathcal{A}$ starting from state i and going to some state m, with the 1-weighted paths of $\mathcal{A}$ starting from that state m and ending up in state j. Since m ranges over all the possible states, these concatenations are in fact *all* the possible 2-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^2$ will consists of all the 2-weighted paths of $\mathcal{A}$ going from an initial to a final state.

Now we want to show that for some $h > 2$, the automaton, say $\mathcal{A}^h_{i,j}$ consists of all the h-weighted paths of $\mathcal{A}$ going from state i to state j. We assume that this is true for all $\mathcal{A}^m_{i,j}$, when $m < h$. Suppose that h is even. The case when h is odd can be similarly dealt with. We have that

$$\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}.$$

From the induction hypothesis, $\mathcal{A}^{h/2}_{i,m}$ and $\mathcal{A}^{h/2}_{m,j}$ consists of *all* the $h/2$-weighted paths of $\mathcal{A}$ going from state i to some state m, and *all* the $h/2$-weighted paths of $\mathcal{A}$ going from that state m to state j respectively. Since m ranges over all the possible states, from the above equation we get that $\mathcal{A}^h_{i,j}$ consists of *all* the possible h-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^h$ will consists of all the h-weighted paths of $\mathcal{A}$ going from an initial to a final state.

□

Notably, writing $\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}$ (supposing h is even) instead of naively writing equivalently $\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h-1}_{i,m} \bullet \mathcal{A}^1_{m,j}$, makes us very efficient with respect to h (and in turn with respect to k) for computing $\mathcal{A}^h_{i,j}$ (and in turn $\mathcal{A}^k_{i,j}$). In order to see that, suppose for simplicity that h is a power of 2. Now, from our equation $\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}$ we have that $\mathcal{A}^2_{i,j}$ will be a union of n automata of size $2p$ (where p is a polynomial on n), $\mathcal{A}^4_{i,j}$ will be a union of n automata of size $4np$, $\mathcal{A}^8_{i,j}$ will be a union of n automata of size $8n^2p$, and so on. Hence, by using our recurrence equation we will get a resulting automaton $\mathcal{A}^h_{i,j}$, which is a union of n automata of length $hn^{log_2 h - 1}p$, i.e. the size of $\mathcal{A}^h_{i,j}$ will be $hn^{log_2 h}p$. In other words, had we used the equivalent equation $\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h-1}_{i,m} \bullet \mathcal{A}^1_{m,j}$, the automata $\mathcal{A}^h_{i,j}$ would be a union of n automata of size pn^{h-1}, i.e. the total size would be pn^h.

We are now ready to show the following theorem.

Theorem 5.2. *The k-limitedness problem is in PSPACE with respect to the size of the automaton. Furthermore, the decision can be made in space sub-exponential with respect to k.*

oarieli@mta.ac.il

PROOF. Recall that to decide whether $d(\mathcal{A}) \leq k$, amounts to testing the language equality $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$. Now, from the above discussion it is clear that the size of $\mathbf{k}(\mathcal{A})$ is $\mathcal{O}(k^2 n^{log_2 k})$. So, we can test the language equivalence $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$ in polynomial space on the size of $\mathcal{A}$ (see [HRS76]), and in sub-exponential space on k. □

Based on the above Theorem and on Theorem 4.3, we can state the following corollary.

Corollary 5.1. *The problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is in PSPACE with respect to the combined size of Q_1, Q_2, and $\mathcal{T}$. Furthermore, the decision can be done in space sub-exponential with respect to k.*

We turn now on the lower bound for deciding the k-containment.

Theorem 5.3. *The problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is PSPACE-hard, even if we know that $Q_1 \subseteq Q_2$.*

PROOF. First recall from Theorem 4.2 that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is equivalent to $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. We will now reduce the NFA universality problem to the $d_{\mathcal{T}}(Q_2, Q_1) \leq k$ problem. The universality problem says: given an NFA $\mathcal{A}$, is $\Delta^* \subseteq L(\mathcal{A})$? The universality problem is PSPACE complete [HRS76].

For an an arbitrary NFA $\mathcal{A}$ on Δ we take $Q_1 = L(\mathcal{A})$. We choose $Q_2 = \Delta^*$ and we take as a distortion transducer $\mathcal{T}$ the one that corresponds to the free applications of the three edit operations insertion, deletion, and substitution. The transducer will consist of a single state which will be both initial and final and loop transitions to this single state. Formally, this transducer is $\mathcal{T} = (\{p\}, \Delta, \tau, \{p\}, \{p\})$, where the transition relation is

$$\begin{aligned}\tau = &\{(p, R, R, 0, p) : R \in \Delta\} \cup \\ &\{(p, \epsilon, R, 1, p) : R \in \Delta\} \cup \\ &\{(p, R, \epsilon, 1, p) : R \in \Delta\} \cup \\ &\{(p, R, S, 1, p) : \{R, S\} \subset \Delta \text{ and } S \neq R\}.\end{aligned}$$

Intuitively, the above says: For each symbol R in Δ we will have a transition R/R leaving the symbol unchanged at no cost, or, at cost 1, through the transitions ϵ/R and R/ϵ we can insert and delete respectively a symbol or finally we can substitute at cost 1 a symbol by another through R/S transitions. Clearly, through the edit distance transducer any word can be transformed (or distorted) to any other word. From this fact, we have that $\Delta^* \subseteq \mathcal{T}(L(\mathcal{A}))$. However, a word can be transformed to another different word only through the application of non-zero cost edit operations. Hence, $d_{\mathcal{T}}(Q_1, Q_2) \leq 0$ if and only if $\Delta^* \subseteq L(\mathcal{A})$. □

Finally, Corollary 5.1 and Theorem 5.3 imply

Corollary 5.2. *To problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is PSPACE complete with respect to the combined size of Q_1, Q_2, and $\mathcal{T}$.*

oarieli@mta.ac.il

6 Deciding Reliable Containment

As we already mentioned in Section 4, the problem of limitedness for distance automata (ϵ-free weighted automata) has been thoroughly investigated by [Has82,Has90,Has00,Leu91,Sim94]. However, in all previous works, the requirement of ϵ-freeness is very essential in devising algorithms for deciding the limitedness of automaton distance.

In this section, we show how to efficiently transform the transducer $\mathcal{C}$ into one with ϵ-free output transitions, is such a way that the essential features of $\mathcal{C}$ are preserved. As a consequence, the automaton $\mathcal{A}$, that we obtain from this output ϵ-free transducer, will be ϵ-free as well.

From the transducer $\mathcal{C}$ we will construct another "distance equivalent" transducer $\mathcal{D}$. We shall use $\epsilon\text{-}closure_{\mathcal{C}}(p)$, similarly to [HU79], to denote the set of all vertices q such that there is path π, from p to q in $\mathcal{C}$, with $out(\pi) = \epsilon$.

Obviously, we will keep all the transitions with non-ϵ output of $\mathcal{C}$ in the transducer $\mathcal{D}$, that we are constructing.

Now, we will insert a transition with R-output ($R \neq \epsilon$) in $\mathcal{D}$ from a state p to a state q whenever there is in $\mathcal{C}$ a path π, with $out(\pi) = \epsilon$, from p to an intermediate state r and there is a transition with R-output, from that state r to the state q. Formally, if $\mathcal{C} = (P, \Delta, \tau, P_0, F)$, then $\mathcal{D} = (P, \Delta, \rho, P_0, G)$, where

$$G = F \cup \{p : p \in P_0 \text{ and } \epsilon\text{-}closure_{\mathcal{C}}(p) \cap F \neq \emptyset\}$$

and

$$\rho = \{(p, R, S, k, q) : (p, R, S, k, q) \in \tau \text{ and } R \neq \epsilon\} \cup \{(p, w, S, \ell, q) : S \neq \epsilon, \exists r \in \epsilon\text{-}closure_{\mathcal{C}}(p), \text{ such that } (r, R, S, m, q) \in \tau\},$$

where w will be a word, such that $w = in(\pi)$, where π is the[5] cheapest path from p to r in $\mathcal{C}$, such that $out(\pi) = \epsilon$. Also, the weight ℓ will be the weight of π (going from p to r) plus the weight of the[6] cheapest transition with R-output, from state r to state q in $\mathcal{C}$.

It is easy to verify about the above constructed transducer $\mathcal{D}$ that

Lemma 6.1. *$rel(\mathcal{D}) \subseteq rel(\mathcal{C})$, $dom(\mathcal{D}) \subseteq dom(\mathcal{C})$, and $ran(\mathcal{D}) = ran(\mathcal{C})$.*

Then, we show that the distance features of $\mathcal{C}$ are preserved in $\mathcal{D}$.

Lemma 6.2. *Let $w \in ran(\mathcal{C}) = ran(\mathcal{D})$. Then $d_{\mathcal{C}}(dom(\mathcal{C}), w) = d_{\mathcal{D}}(dom(\mathcal{D}), w)$*

Proof. We have that

$$d_{\mathcal{C}}(dom(\mathcal{C}), w) = inf\{d_{\mathcal{C}}(u, w) : u \in dom(\mathcal{C})\}.$$

Let $u_0 \in dom(\mathcal{C})$ such that $d_{\mathcal{C}}(u_0, w) = d_{\mathcal{C}}(dom(\mathcal{C}), w)$, and consider the corresponding cheapest path π, labeled u_0/w, in $\mathcal{C}$. For simplicity suppose that u_0 is

[5] The cheapest path can be non-unique.
[6] The cheapest transition can also be non-unique.

oarieli@mta.ac.il

unique. The case when u_0 is not unique can be handled with some additional straightforward omitted technicalities. If none of the edges of π is labeled with ϵ as output, then by the construction we know that the path π is also in $\mathcal{D}$ and the proof is finished.

Now, let's suppose that some part of the path π is

$$p_1 \overset{R_1/\epsilon,\, h_1}{\longrightarrow} p_2, \ldots, p_{k-1} \overset{R_{k-1}/\epsilon,\, h_{k-1}}{\longrightarrow} p_k, p_k \overset{R_k/S,\, h_k}{\longrightarrow} p_{k+1},$$

where $S \neq \epsilon$.

We claim that in $\mathcal{D}$ we will have the transition $(p_1, R_1 \ldots R_k, S, h, p_{k+1})$ with weight $h = h_1 + \cdots + h_k$.

The only way that this could not be true is, if in $\mathcal{C}$ there is a path, comprised of ϵ-output transitions, which is cheaper than using π, of getting from p_1 to p_k. Suppose this cheap path is labeled $T_1/\epsilon, \ldots, T_{m-1}/\epsilon$. There could also be a transition from p_k to p_{k+1}, labeled with T_m/S, that is cheaper than the last π-transition. If such a cheaper path were to exist, and if we think of u_0 as being of the form $xR_1 \ldots R_{k-1}R_k y$, then for the word $u_1 = xT_1 \ldots T_{m-1}T_m y$, we would have $d_{\mathcal{C}}(u_1, w) < d_{\mathcal{C}}(u_0, w)$, which is a contradiction. Hence $(p_1, R_1 \ldots R_k, S, p_{k+1}, h)$ belongs to the transition relation of $\mathcal{D}$. Now, we can decompose the path π into disjoint (with respect to edges) subpaths such as the above, and subpaths that do not have any edge labeled by ϵ on the output. Recall that the subpaths that do not have any edge labeled by ϵ on the output are kept in $\mathcal{D}$, and so finally, we have that there exists a path σ in $\mathcal{D}$ labeled u_0/w and with the same weight as π. From this, we conclude that $d_{\mathcal{C}}(dom(\mathcal{C}), w) \geq d_{\mathcal{D}}(dom(\mathcal{D}), w)$.

To prove $d_{\mathcal{C}}(dom(\mathcal{C}), w) \leq d_{\mathcal{D}}(dom(\mathcal{D}), w)$ we reason in the following way. From the Lemma 6.1, we have that $rel(\mathcal{D}) \subseteq rel(\mathcal{C})$. So, if $(u, w) \in rel(\mathcal{D})$ then $(u, w) \in rel(\mathcal{C})$ and by the construction of $\mathcal{D}$ we know that for the cheapest path in $\mathcal{D}$ labeled with u/w there is a corresponding path in $\mathcal{D}$ labeled with u/w and having the same weight. Based on this observation the above inequality follows. □

If we now eliminate the input from the transitions in $\mathcal{D} = (P, \Delta, \rho, P_0, G)$, we obtain an ϵ-free distance automaton $\mathcal{A} = (P, \Delta, \tau_{\mathcal{A}}, P_0, G)$, where

$$\tau_{\mathcal{A}} = \{(p, R, \ell, q) : (p, w, R, k, q) \in \rho \text{ for some } w\},$$

and the weight ℓ is given by the weight of the (possibly non-unique) corresponding cheapest transition in the transducer $\mathcal{D}$, i.e.

$$\ell = inf\{k : (p, w, R, k, q) \in \rho \text{ for some } w\}.$$

Now, we can state the following theorem.

Theorem 6.1. *Let Q_1 and Q_2 be queries and $\mathcal{T}$ a distortion transducer. Compute the output ϵ-free Cartesian product transducer $\mathcal{D}$ and consider the distance automaton $\mathcal{A}$ constructed from $\mathcal{A}$. Then, $d(\mathcal{A}) = d_{\mathcal{T}}(Q_1, Q_2)$.*

oarieli@mta.ac.il

PROOF. For $p \in P_0$ and $q \in G$, let w be a word in Q_2 such that there exists a path π between the states p and q in $\mathcal{D}$ spelling w as output, i.e. $w = out(\pi)$. By the construction of $\mathcal{D}$ and $\mathcal{A}$, we have that $d_{\mathcal{A}}(p, w, q) = d_{\mathcal{D}}(dom(\mathcal{D}), w)$. Now, by using Lemma 6.2 and Lemma 4.1 we derive

$$\begin{aligned} d_{\mathcal{A}}(p, w, q) &= d_{\mathcal{D}}(dom(\mathcal{D}), w) \\ &= d_{\mathcal{C}}(dom(\mathcal{C}), w) \\ &= d_{\mathcal{T}}(Q_1, w). \end{aligned}$$

Since w was an arbitrary word in Q_2, we finally get that $d(\mathcal{A}) = d_{\mathcal{T}}(Q_1, Q_2)$. □

Hence, we are able now to use Leung's algorithm [Leu91], which is computationally the best known algorithm for solving the limitedness problem (in single exponential time), but for which the ϵ-freeness of the automata is essential.

Additionally, we show the following complexity bound for the reliable containment, which says that the exponential time algorithm of Leung is almost the best one could do for deciding the problem of reliable containment.

Theorem 6.2. *The reliable query containment problem is PSPACE-hard.*

PROOF. We will give a reverse reduction from the limitedness problem, which is known to be PSPACE-hard [Leu91].

Let $\mathcal{A} = (P, \Delta, \tau, P_0, F)$ be a distance automaton. We take another alphabet Γ of the same size as, but disjoint from Δ. Let φ be a one-to-one mapping from Δ onto Γ. Now, from the automaton $\mathcal{A}$ we construct a distortion transducer $\mathcal{T} = (P \cup \{p_{id}\}, \Delta \cup \Gamma, \rho, P_0 \cup \{p_{id}\}, F)$, where

$$\begin{aligned} \rho = &\{(p, \varphi(R), R, k, q) : (p, R, k, q) \in \tau\} \cup \\ &\{(p_{id}, R, R, 0, p_{id}) : R \in \Delta \cup \Gamma\}. \end{aligned}$$

Note that by adding the new state p_{id} in the above transducer, we do have the property to "leave everything" unchanged that is required from a distortion transducer, in order include the original query words, undistorted, in the result of transduction.

Let $\mathcal{U}$ be the transducer that get if we drop in $\mathcal{T}$ the state p_{id}, and let $\mathcal{V}$ be the identity transducer that get if we drop all the other states and keep p_{id}. Clearly, the transducer $\mathcal{T}$ can be seen as the union of $\mathcal{U}$ and $\mathcal{V}$.

Now, let $Q_1 = dom(\mathcal{U})$ and $Q_2 = ran(\mathcal{U}) = L(\mathcal{A})$. Since, $\Gamma \cap \Delta = \emptyset$ and $Q_1 \subseteq \Gamma^*$, while $Q_2 \subseteq \Delta^*$, we have that $Q_1 \cap Q_2 = \emptyset$. So, we cannot get any word of Q_2 from Q_1 through the 0-weighted identity transducer $\mathcal{V}$, i.e. Q_1 can be distorted only through $\mathcal{U}$ (which closely corresponds to $\mathcal{A}$) to Q_2.

Finally, from all the above we have that $d_{\mathcal{T}}(Q_1, Q_2) = d_{\mathcal{U}}(Q_1, Q_2) = d(\mathcal{A})$. Hence, $d(\mathcal{A}) \leq k$ if $d_{\mathcal{T}}(Q_1, Q_2) \leq k$, which by Theorem 4.2 is equivalent with $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$. □

oarieli@mta.ac.il

References

[ABS99] Abiteboul S., P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and Xml.* Morgan Kaufmann Pulishers. San Francisco, Ca., 1999.

[AHU74] Aho A., J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley. Reading Ma., 1974.

[C+99] Calvanese D., G. Giacomo, M. Lenzerini and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Proc. PODS '99*, pp. 194–204.

[C+00] Calvanese D., G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing and Constraint Satisfaction. *Proc. LICS '00*, pp. 361–371

[GT00] Grahne G., and A. Thomo. An Optimization Technique for Answering Regular Path Queries *Proc. WebDB '00*, pp. 99–104.

[GT01] Grahne G., and A. Thomo. Algebraic rewritings for optimizing regular path queries. *Proc. ICDT '01*, pp. 303–315

[Has82] Hashiguchi K. Limitedness Theorem on Finite Automata with Distance Functions. *J. Comp. Syst. Sci.* 24 (2) : 233–244, 1982

[Has90] Hashiguchi K. Improved Limitedness Theorems on Finite Automata with Distance Functions. *Theoretical Computer Science* 72 (1) : 27–38, 1990

[Has00] Hashiguchi K. New upper bounds to the limitedness of distance automata. *Theoretical Computer Science* 233 (1-2) : 19–32, 2000

[HU79] Hopcroft J. E., and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley. Reading Ma., 1979.

[HRS76] Hunt H. B. III, D. J. Rosenkrantz, and T. G. Szymanski, On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages. *J. Comp. Syst. Sci.* 12 (2) : 222–268, 1976

[Kru83] Kruskal J. An Overview of Sequence Comparison. In: *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* D. Sankoff and J. Kruskal (Eds.). Addison-Wesley. Reading Ma., 1983. pp. 1–44.

[JMM95] Jagadish H. V., A. O. Mendelzon, and T. Milo. Similarity-Based Queries. *Proc. PODS '95*, pp. 36–45.

[Leu91] Leung H. Limitedness Theorem on Finite Automata with Distance Functions: An Algebraic Proof. *Theoretical Computer Science* 81 (1) : 137–145, 1991

[MW95] Mendelzon A. O., and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp.* 24 (6) : 1235–1258, 1995.

[MMM97] Mendelzon A. O. G. A. Mihaila and T. Milo. Querying the World Wide Web. *Int. J. Dig. Lib.* 1 (1) : 57–67, 1997

[Pin98] Pin. J. E. Tropical Semirings, *in Idempotency*, J. Gunawardena (ed.) Cambridge University Press, pp. 50–69, 1998

[Sim94] Simon I. On Semigroups of Matrices over the Tropical Semiring. *Informatique Theorique et Applications* 28 (3-4) : 277–294, 1994

[WF74] Wagner R. A., and M. J. Fischer. The String-to-String Correction Problem. *J. ACM* 21 (1) : 168–173, 1974

oarieli@mta.ac.il

Weak Functional Dependencies in Higher-Order Datamodels

– The Case of the Union Constructor –

Sven Hartmann, Sebastian Link, and Klaus-Dieter Schewe

Massey University, Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
[s.hartmann|s.link|k.d.schewe]@massey.ac.nz

Abstract. We present an axiomatisation for weak functional dependencies, i.e. disjunctions of functional dependencies, in the presence of several constructors for complex values. These constructors are the tuple constructor, the set-constructor, an optionality constructor, and a union constructor. The theory is smooth and rather uniform, if the union-constructor is absent. Its presence, however, complicates all results and proofs significantly. The reason for this is that the union-constructor comes alomg with non-trivial restructuring rules. In particular, if the union-constructor is absent, a subset of the rules is complete for the implication of ordinary functional dependencies, but this does not hold, if the union constructor is present.

1 Introduction

In the relational datamodel (RDM) a lot of research has been spent on the theory of dependencies, i.e. first-order sentences that are supposed to hold for all database instances [3,19]. Various classes of dependencies for the RDM have been introduced [22], and large parts of database theory deals with the finite axiomatisation of these dependencies and the finite implication problem for them. That is to decide that a dependency φ is implied by a set of dependencies Σ, where implication refers to the fact that all finite models of Σ are also models of φ. The easiest, yet most important class of dependencies is the class of functional dependencies. Armstrong [5] was the first to give a finite axiomatisation for FDs.

Dependency theory is a cornerstone of database design, as the semantics of the application domain cannot be expressed only by structures. Database theory has to investigate the implications arising from the presence of dependencies. This means to describe semantically desirable properties of "well-designed" databases, e.g., the absence of redundancy, to characterise them (if possible) syntactically by in-depth investigation of the dependencies and to develop algorithms to transform schemata into normal forms, which guarantee the desirable properties to be satisfied.

However, the field of databases is no longer the unique realm of the RDM. First, so called semantic datamodels have been developed [8,16], which were

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 116–133, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

originally just meant to be used as design aids, as application semantics was assumed to be easier captured by these models [6,9,25]. Later on some of these models, especially the nested relational model [19], object oriented models [20] and object-relational models, the gist of which are captured by the higher-order Entity-Relationship model (HERM, [23,24]) have become interesting as datamodels in their own right and some dependency and normalisation theory has been carried over to these advanced datamodels [12,17,18,19,21]. Most recently, the major research interest is on the model of semi-structured data and XML [1], which may also be regarded as some kind of object oriented model.

We refer to all these models as "higher-order" datamodels. This is, because the most important extension that came with these models was the introduction of constructors for complex values. These constructors usually comprise bulk constructors for sets, lists and multisets, a (disjoint-)union constructor, and an optionality or null-constructor. In fact, all the structure of higher-order datamodels (including XML as far as XML can be considered a datamodel) is captured by the introduction of (some or all of) these constructors.

The key problem is to develop dependency theories (or preferably a unified theory) for the higher-order datamodels. The development of such a dependency theory will have a significant impact on understanding application semantics and laying the grounds for a logically founded theory of well-designed non-relational databases.

So far, mainly keys and FDs for advanced datamodels have been investigated [7,11,10] and led to several normal form proposals [4,15]. Only the work in [15] contains explicit definitions of redundancy and update anomalies and proves (in the spirit of the work in [26]) that the suggested higher-level normal form (HLNF) in the presence of FDs is indeed equivalent to the absence of redundancy and sufficient for the absence of update anomalies. Another difference is that the work in [4] tries to reduce the problem to dependencies arising from a relational representation of XML documents, thus (similar to [17]) restricts the attention to those FDs that arise from the relational representation, whereas our work addresses the problem in the context of types for nested attributes and subtyping.

So far our work on functional dependencies exploited the set-constructor only. For this case an axiomatisation was presented in [14,13]. Particular care had to taken for a generalisation of the extension rule, which does no longer hold in general. In this article we continue our work and investigate the combination with the union-constructor, which will increase the complexity of the problem tremendously. We have to take restructuring rules into consideration and extent the order on equivalence classes of attributes. Then we achieve an axiomatisation for weak functional dependencies, but this axiomatisation contains additional axioms. Surprisingly, the completeness proof for functional dependencies only, i.e. without disjunctions, cannot be generalised. Thus, in order to derive all implied functional dependencies we have to reason with disjunctions.

In Section 2 we briefly define our abstract model of attributes and investigate the structure of the set of subattributes of a given nested attribute. We obtain a lattice with a relative pseudo-complement operation, but we lose the distribu-

oarieli@mta.ac.il

tivity, i.e. we do not get a full Brouwer algebra. Then in Section 3 we present sound axioms and rules for functional and weak functional dependencies. Before we can approach the completeness of these rules in Section 5 we need technical results on ideals and *strong higher-level ideals* (SHL-ideals, Section 4), i.e. ideals with additional closure properties. The main result of the technical Section 4 is the existence of values that coincide exactly on a given SHL-ideal. We refer to this result as the Central Lemma. The Central Lemma will be the most important tool in the completeness proof. Due to limited space all proofs that are only technical and use boring structural induction had to be omitted.

2 Algebras of Nested Attributes

In this section we define our model of nested attributes, which covers the gist of higher-order datamodels including XML. In particular, we investigate the structure of the set $\mathcal{S}(X)$ of subattributes of a given nested attribute X. We show that we obtain a non-distributive Brouwer algebra, i.e. a non-distributive lattice with relative pseudo-complements.

2.1 Nested Attributes

We start with a definition of simple attributes and values for them.

Definition 1. A *universe* is a finite set $\mathcal{U}$ together with domains (i.e. sets of values) $dom(A)$ for all $A \in \mathcal{U}$. The elements of $\mathcal{U}$ are called *simple attributes*.

For the relational model a universe was enough, as a relation schema could be defined by a subset $R \subseteq \mathcal{U}$. For higher-order datamodels, however, we need nested attributes. In the following definition we use a set $\mathcal{L}$ of labels, and tacitly assume that the symbol λ is neither a simple attribute nor a label, i.e. $\lambda \notin \mathcal{U} \cup \mathcal{L}$, and that simple attributes and labels are pairwise different, i.e. $\mathcal{U} \cap \mathcal{L} = \emptyset$.

Definition 2. Let $\mathcal{U}$ be a universe and $\mathcal{L}$ a set of labels. The set $\mathcal{N}$ of *nested attributes* (over $\mathcal{U}$ and $\mathcal{L}$) is the smallest set with $\lambda \in \mathcal{N}$, $\mathcal{U} \subseteq \mathcal{N}$, and satisfying the following properties:

- for $X \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X(X'_1, \ldots, X'_n) \in \mathcal{N}$;
- for $X \in \mathcal{L}$ and $X' \in \mathcal{N}$ we have $X\{X'\} \in \mathcal{N}$;
- for $X_1, \ldots, X_n \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \in \mathcal{N}$.

We call λ a *null attribute*, $X(X'_1, \ldots, X'_n)$ a *record attribute*, $X\{X'\}$ a *set attribute*, and $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ a *union attribute*. As record and set attributes have a unique leading label, say X, we often write simply X to denote the attribute.

We can now extend the association *dom* from simple to nested attributes, i.e. for each $X \in \mathcal{N}$ we will define a set of values $dom(X)$.

oarieli@mta.ac.il

Definition 3. For each nested attribute $X \in \mathcal{N}$ we get a *domain* $dom(X)$ as follows:

- $dom(\lambda) = \{\top\}$;
- $dom(X(X'_1, \ldots, X'_n)) = \{(X_1 : v_1, \ldots, X_n : v_n) \mid v_i \in dom(X'_i)$ for $i = 1, \ldots, n\}$ with labels X_i for the attributes X'_i;
- $dom(X\{X'\}) = \{\{v_1, \ldots, v_n\} \mid v_i \in dom(X')$ for $i = 1, \ldots, n\}$, i.e. each element in $dom(X\{X'\})$ is a finite set with elements in $dom(X')$;
- $dom(X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)) = \{(X_i : v_i) \mid v_i \in dom(X'_i)$ for $i = 1, \ldots, n\}$.

Note that the relational model is covered, if only the tuple constructor is used. Thus, instead of a relation schema R we will now consider a nested attribute X, assuming that the universe $\mathcal{U}$ and the set of labels $\mathcal{L}$ are fixed. Instead of an R-relation r we will consider a finite set $r \subseteq dom(X)$.

2.2 Subattributes

In the dependency theory for the relational model we considered the powerset $\mathcal{P}(R)$ for a relation schema R. $\mathcal{P}(R)$ is a Boolean algebra with order $\subseteq$, intersection $\cap$, union $\cup$ and the difference $-$.

We will generalise these operations for nested attributes starting with a partial order $\geq$. However, this partial order will be defined on equivalence classes of attributes. We will identify nested attributes, if we can identify their domains.

Definition 4. $\equiv$ is the smallest *equivalence relation* on $\mathcal{N}$ satisfying the following properties:

- $\lambda \equiv X()$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_1, \ldots, X'_n, \lambda)$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_{\sigma(1)}, \ldots, X'_{\sigma(n)})$ for any permutation σ;
- $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \equiv X_1(X'_{\sigma(1)}) \oplus \cdots \oplus X_n(X'_{\sigma(n)})$ for any permutation σ;
- $X(X'_1, \ldots, X'_n) \equiv X(Y_1, \ldots, Y_n)$ iff $X'_i \equiv Y_i$ for all $i = 1, \ldots, n$;
- $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \equiv X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ iff $X'_i \equiv Y_i$ for all $i = 1, \ldots, n$;
- $X\{X'\} \equiv X\{Y\}$ iff $X' \equiv Y$;
- $X(X'_1, \ldots, Y_1(Y'_1) \oplus \cdots \oplus Y_m(Y'_m), \ldots, X'_n) \equiv Y_1(X'_1, \ldots, Y'_1, \ldots, X'_n) \oplus \cdots \oplus Y_m(X'_1, \ldots, Y'_m, \ldots, X'_n)$;
- $X\{X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)\} \equiv X(X_1\{X'_1\}, \ldots, X_n\{X'_n\})$.

Basically, the equivalence definition (apart from the last three cases) states that λ in record attributes can be added or removed, and that order in record and union attributes does not matter. The last two cases in Definition 4 cover restructuring rules that were already introduced in [2]. Obviously, if we have a set of labelled elements, we can split this set into n subsets, each of which contains just the elements with a particular label, and the union of these sets is the original set.

In the following we identify $\mathcal{N}$ with the set $\mathcal{N}/_{\equiv}$ of equivalence classes. In particular, we will write $=$ instead of $\equiv$, and in the following definition we should say that Y is a subattribute of X iff $\tilde{X} \geq \tilde{Y}$ holds for some $\tilde{X} \equiv X$ and $\tilde{Y} \equiv Y$.

oarieli@mta.ac.il

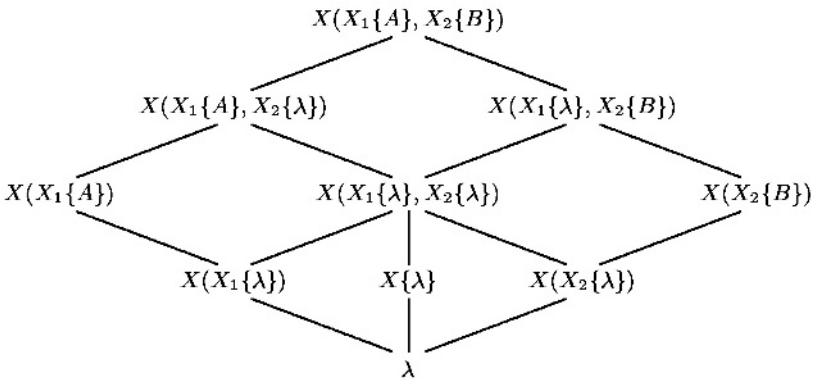


Fig. 1. The lattice $\mathcal{S}(X\{X_1(A) \oplus X_2(B)\})$

Definition 5. For $X, Y \in \mathcal{N}$ we say that Y is a *subattribute* of X, iff $X \geq Y$ holds, where $\geq$ is the smallest partial order on $\mathcal{N}$ satisfying the following properties:

- $X \geq \lambda$ for all $X \in \mathcal{N}$;
- $X(Y_1, \ldots, Y_n) \geq X(X'_{\sigma(1)}, \ldots, X'_{\sigma(m)})$ for some injective $\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $Y_{\sigma(i)} \geq X'_{\sigma(i)}$ for all $i = 1, \ldots, m$;
- $X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \geq X_{\sigma(1)}(X'_{\sigma(1)}) \oplus \cdots \oplus X_{\sigma(n)}(X'_{\sigma(n)})$ for some permutation σ and $Y_i \geq X'_i$ for all $i = 1, \ldots, n$;
- $X\{Y\} \geq X\{X'\}$ iff $Y \geq X'$;
- $X(X_{i_1}\{\lambda\}, \ldots, X_{i_k}\{\lambda\}) \geq X_{\{i_1, \ldots, i_k\}}\{\lambda\}$.

Note that the last case in Definition 5 arises from the restructuring for sets of unions. We have to add a little remark on notation here. As we identify $X\{X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)\}$ with $X(X_1\{X'_1\}, \ldots, X_n\{X'_n\})$, we obtain subattributes $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})$ for each subset $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. As we can also indentify such a subattribute with $X\{X_{i_1}(X'_{i_1}) \oplus \cdots \oplus X_{i_k}(X'_{i_k})\}$, we obtain subattributes of the form "$X\{\lambda\}$" for all such subsets of indices. In order to distinguish these subattributes from each other, we use new labels X_I and write $X_I\{\lambda\}$. Note that $X_\emptyset\{\lambda\} = \lambda$, $X_{\{1,\ldots,n\}}\{\lambda\} = X\{\lambda\}$ and $X_{\{i\}}\{\lambda\} = X(X_i\{\lambda\})$.

Further note that due to the restructuring rules in Definitions 4 and 5 we may have the case that a record attribute is a subattribute of a set attribute and vice versa. This cannot be the case, if the union-constructor is absent. However, the presence of the restructuring rules allows us to assume that the union-constructor only appears inside a set-constructor or as the outermost constructor. This will be frequently exploited in our proofs.

Obviously, $X \geq Y$ induces a projection map $\pi^X_Y : dom(X) \to dom(Y)$. For $X \equiv Y$ we have $X \geq Y$ and $Y \geq X$ and the projection maps π^X_Y and π^Y_X are inverse to each other.

oarieli@mta.ac.il

We use the notation $\mathcal{S}(X) = \{Z \in \mathcal{N} \mid X \geq Z\}$ to denote the *set of subattributes* of a nested attribute X. Figure 1 shows the subattributes of $X\{X_1(A) \oplus X_2(B)\} = X(X_1\{A\}, X_2\{B\})$ together with the relation $\geq$ on them. Note that the subattribute $X\{\lambda\}$ would not occur, if we only considered the record-structure, whereas other subattributes such as $X(X_1\{\lambda\})$ would not occur, if we only considered the set-structure. This is a direct consequence of the restructuring rules.

2.3 The Brouwer Algebra Structure

Let us now investigate the structure of $\mathcal{S}(X)$. We obtain a non-distributive Brouwer algebra, i.e. a non-distributive lattice with relative pseudo-complements.

Definition 6. Let $\mathcal{L}$ be a lattice with zero and one, partial order $\leq$, join $\sqcup$ and meet $\sqcap$. $\mathcal{L}$ has *relative pseudo-complements* iff for all $Y, Z \in \mathcal{L}$ the infimum $Y \leftarrow Z = \bigsqcap\{U \mid U \sqcup Y \geq Z\}$ exists. Then $Y \leftarrow 1$ (1 being the one in $\mathcal{L}$) is called the *relative complement* of Y.

If we have distributivity in addition, we call $\mathcal{L}$ a *Brouwer algebra*. In this case the relative pseudo-complements satisfy $U \geq (Y \leftarrow Z)$ iff $(U \sqcup Y \geq Z)$, but if we do not have distributivity this property may be violated though relative pseudo-complements exist.

Proposition 1. *The set $\mathcal{S}(X)$ of subattributes carries is structure of a lattice with zero and one and relative pseudo-complements, where the order $\geq$ is as defined in Definition 5, and λ and X are the zero and one, respectively.*

It is easy to determine explicit inductive definitions of the operations $\sqcap$ (meet), $\sqcup$ (join) and $\leftarrow$ (relative pseudo-complement). This can be done by boring technical verification of the properties of meets, joins and relative pseudo-complements and is therefore omitted here.

Example 1. Let $X = X\{X_1(A) \oplus X_2(B)\}$ with $\mathcal{S}(X)$ as illustrated in Figure 1, $Y_1 = X\{\lambda\}$, $Y_2 = X(X_2\{B\})$, and $Z = X(X_1\{A\}$. Then we have

$$Z \sqcap (Y_1 \sqcup Y_2) = X(X_1\{A\} \sqcap (X\{\lambda\} \sqcup X(X_2\{B\})) =$$

$$X(X_1\{A\}) \sqcap X(X_1\{\lambda\}, X_2\{B\}) = X(X_1\{\lambda\}) \neq \lambda = \lambda \sqcup \lambda =$$

$$(X(X_1\{A\}) \sqcap X\{\lambda\}) \sqcup (X(X_1\{A\}) \sqcap X(X_2\{B\})) = (Z \sqcap Y_1) \sqcup (Z \sqcap Y_2)\ .$$

This shows that $\mathcal{S}(X)$ in general is not a distributive lattice. Furthermore, $Y' \sqcup Z \geq Y_1$ holds for all Y' except λ, $X(X_1\{\lambda\})$ and $X(X_1\{A\})$. So $Z \leftarrow Y_1 = \lambda$, but not all $Y' \geq \lambda$ satisfy $Y' \sqcup Z \geq Y_1$. □

oarieli@mta.ac.il

3 Sound Derivation Rules for Weak Functional Dependencies

In this section we will define functional and weak functional dependencies on $\mathcal{S}(X)$ and derive some sound derivation rules. The first thought would be to consider single nested attributes, as in the RDM $\sqcup$ corresponds to the union $\cup$, and $\sqcap$ to the intersection $\cap$. However, if we treat functional dependencies in this way, we cannot obtain a generalisation of the extension rule. Therefore, we have to consider sets of subattributes.

Definition 7. Let $X \in \mathcal{N}$. A *functional dependency* (FD) on $\mathcal{S}(X)$ is an expression $\mathcal{Y} \to \mathcal{Z}$ with $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X)$. A *weak functional dependency* (wFD) on $\mathcal{S}(X)$ is an expression $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with an index set I and $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X)$.

In the following we consider finite sets $r \subseteq dom(X)$, which we will call simply *instances* of X.

Definition 8. Let r be an instance of X. We say that r *satisfies the FD* $\mathcal{Y} \to \mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models \mathcal{Y} \to \mathcal{Z}$) iff for all $t_1, t_2 \in r$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ we also have $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$.

r *satisfies the wFD* $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ on $\mathcal{S}(X)$ (notation: $r \models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$) iff for all $t_1, t_2 \in r$ there is some $i \in I$ with $\{t_1, t_2\} \models \mathcal{Y}_i \to \mathcal{Z}_i$.

According to this definition we identify a wfD $\{\!\{\mathcal{Y} \to \mathcal{Z}\}\!\}$, i.e. the index set contains exactly one element, with the "ordinary" FD $\mathcal{Y} \to \mathcal{Z}$.

Let Σ be a set of FDs and wFDs. A FD or wFD ψ is implied by Σ (notation: $\Sigma \models \psi$) iff all instances r with $r \models \varphi$ for all $\varphi \in \Sigma$ also satisfy ψ. As usual we write $\Sigma^* = \{\psi \mid \Sigma \models \psi\}$.

As usual we write Σ^+ for the set of all FDs and wFDs that can be derived from Σ by applying a system $\mathfrak{R}$ of axioms and rules, i.e. $\Sigma^+ = \{\psi \mid \Sigma \vdash_{\mathfrak{R}} \psi\}$. We omit the standard definitions of derivations with a given rule system, and also write simply $\vdash$ instead of $\vdash_{\mathfrak{R}}$, if the rule system is clear from the context.

Our goal is to find a finite axiomatisation, i.e. a rule system $\mathfrak{R}$ such that $\Sigma^* = \Sigma^+$ holds. The rules in $\mathfrak{R}$ are *sound* iff $\Sigma^+ \subseteq \Sigma^*$ holds, and *complete* iff $\Sigma^* \subseteq \Sigma^+$ holds.

3.1 Axioms and Rules for Functional Dependencies

Let us first look only at "ordinary" FDs. We indicated above that we cannot obtain a simple generalisation of Armstrong's extension rule for FDs in the relational model, so we will need a particular notion of "semi-disjointness" that will permit such a generalisation.

Definition 9. Two subattributes $Y, Z \in \mathcal{S}(X)$ are called *semi-disjoint* iff one of the following holds:

1. $Y \geq Z$ or $Z \geq Y$;
2. $X = X(X_1, \ldots, X_n)$, $Y = X(Y_1, \ldots, Y_n)$, $Z = X(Z_1, \ldots, Z_n)$ and $Y_i, Z_i \in \mathcal{S}(X_i)$ are semi-disjoint for all $i = 1, \ldots, n$;

oarieli@mta.ac.il

3. $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$, $Y = X_1(Y_1') \oplus \cdots \oplus X_n(Y_n')$, $Z = X_1(Z_1') \oplus \cdots \oplus X_n(Z_n')$ and $Y_i', Z_i' \in \mathcal{S}(X_i')$ are semi-disjoint for all $i = 1, \ldots, n$.

Note that for the set- and multiset-constructor we can only obtain semi-disjointness for subattributes in a $\geq$-relation. With the notion of semi-disjointness we can formulate axioms and rules for FDs and show their soundness.

Theorem 1. *The following axioms and rules are sound for the implication of FDs on* $\mathcal{S}(X)$*:*

λ axiom:
$$\frac{}{\emptyset \to \{\lambda\}} \tag{1}$$

subattribute axiom:
$$\frac{}{\{Y\} \to \{Z\}}\, Y \geq Z \tag{2}$$

join axiom:
$$\frac{}{\{Y, Z\} \to \{Y \sqcup Z\}}\, Y, Z \text{ semi-disjoint} \tag{3}$$

reflexivity axiom:
$$\frac{}{\mathcal{Y} \to \mathcal{Z}}\, \mathcal{Z} \subseteq \mathcal{Y} \tag{4}$$

extension rule:
$$\frac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \mathcal{Y} \cup \mathcal{Z}} \tag{5}$$

transitivity rule:
$$\frac{\mathcal{Y} \to \mathcal{Z} \quad \mathcal{Z} \to \mathcal{U}}{\mathcal{Y} \to \mathcal{U}} \tag{6}$$

set axiom:
$$\frac{}{\{X_I\{\lambda\}, X_J\{\lambda\}\} \to \{X_{I \cup J}\{\lambda\}\}}\, I \cap J = \emptyset \tag{7}$$

set lifting rule:
$$\frac{\{Y\} \to \mathcal{Z}}{\{X\{Y\}\} \to \{X\{Z\} \mid Z \in \mathcal{Z}\}}\, X = X\{X'\},\ Y \in \mathcal{S}(X'), \mathcal{Z} \subseteq \mathcal{S}(X') \tag{8}$$

record lifting rule:
$$\frac{\mathcal{Y}_i \to \mathcal{Z}_i}{\{X(\lambda, \ldots, Y_i, \ldots, \lambda) \mid Y_i \in \mathcal{Y}_i\} \to \{X(\lambda, \ldots, Z_i, \ldots, \lambda) \mid Z_i \in \mathcal{Z}_i\}}\, \mathcal{C}$$
$$\text{with conditions } \mathcal{C} : X = X(X_1, \ldots, X_n) \text{ and } \mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X_i) \tag{9}$$

union lifting rule:
$$\frac{\mathcal{Y}_i \to \mathcal{Z}_i}{\{\cdots \oplus X_i(Y_i) \oplus \cdots \mid Y_i \in \mathcal{Y}_i\} \to \{\cdots \oplus X_i(Z_i) \oplus \cdots \mid Z_i \in \mathcal{Z}_i\}}\, \mathcal{C}$$
$$\text{with conditions } \mathcal{C} : X = X(X_1, \ldots, X_n) \text{ and } \mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X_i'), \mathcal{Y}_i \neq \emptyset \tag{10}$$

We omit the easy proof. Note that (4), (5) and (6) are the Armstrong-axioms that are well-known from the RDM. Axioms (1), (2) and (3) are structural axioms dealing with the Brouwer algebra structure on $\mathcal{S}(X)$. The set-axiom (7) is only needed in the presence of the union-constructor. The same applies to the lifting

oarieli@mta.ac.il

rules (8), (9) and (10). If the union-constructor is absent, we can only "lift" such FDs that can also be derived without using the lifting rules.

It is easy to derive additional rules:

union rule: $$\frac{\mathcal{Y} \to \mathcal{Z} \quad \mathcal{Y} \to \mathcal{U}}{\mathcal{Y} \to \mathcal{Z} \cup \mathcal{U}} \tag{11}$$

fragmentation rule: $$\frac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \{Z\}}\ Z \in \mathcal{Z} \tag{12}$$

join rule: $$\frac{\{Y\} \to \{Z\}}{\{Y\} \to \{Y \sqcup Z\}}\ Y, Z \text{ semi-disjoint} \tag{13}$$

For the union rule (11) we use the following derivation:

$$\dfrac{\dfrac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \mathcal{Y} \cup \mathcal{Z}} \quad \dfrac{\dfrac{\dfrac{}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Y}} \quad \mathcal{Y} \to \mathcal{U}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{U}}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Y} \cup \mathcal{Z} \cup \mathcal{U}} \quad \dfrac{}{\mathcal{Y} \cup \mathcal{Z} \cup \mathcal{U} \to \mathcal{Z} \cup \mathcal{U}}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Z} \cup \mathcal{U}}}{\mathcal{Y} \to \mathcal{Z} \cup \mathcal{U}}$$

For the fragmentation rule (12) we use the following derivation:

$$\dfrac{\mathcal{Y} \to \mathcal{Z} \quad \dfrac{}{\mathcal{Z} \to \{Z\}}}{\mathcal{Y} \to \{Z\}}$$

Finally, for the join-rule (13) we use the following derivation:

$$\dfrac{\dfrac{\{Y\} \to \{Z\}}{\{Y\} \to \{Y, Z\}} \quad \dfrac{}{\{Y, Z\} \to \{Y \sqcup Z\}}}{\{Y\} \to \{Y \sqcup Z\}}$$

3.2 Axioms and Rules for Weak Functional Dependencies

The axioms and rules in Theorem 1 only apply to "ordinary" FDs. For the implication of wFDs we need additional axioms and rules.

Theorem 2. *The following axioms and rules are sound for the implication of wFDs on* $\mathcal{S}(X)$*:*

– *union axiom (for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$ *and* $I = \{i_1, \ldots, i_k\}$*):*

$$\frac{}{\{\{X_I\{\lambda\}\} \to \{X_J\{\lambda\}\}, \{X_I\{\lambda\}\} \to \{X(X_{i_1}\{X_{i_1}'\}, \ldots, X_{i_k}\{X_{i_k}'\})\}\}}\ I \subsetneq J \tag{14}$$

– *partition axioms (for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$ *and* $I \subseteq \{1, \ldots, n\}$*):*

$$\frac{}{\begin{array}{l}\{\{X_I\{\lambda\}\} \to \{X_{I_1' \cup I_2'}\{\lambda\} \mid \emptyset \neq I_1' \subseteq I_1, \emptyset \neq I_2' \subseteq I_2\},\\ \quad \{X_I\{\lambda\}\} \to \{X(X_i\{\lambda\})\} \mid I = I_1 \cup I_2, I_1 \cap I_2 = \emptyset, I_1 \neq \emptyset \neq I_2, i \in I\}\end{array}}$$

oarieli@mta.ac.il

and

$$\frac{}{\{\!\{\{X_{I'}\{\lambda\} \mid I' \cap I_1 \neq \emptyset \neq I' \cap I_2\} \cup \{X_{I_1}\{\lambda\}\} \to \{X(X_i\{\lambda\})\} \mid I = I_1 \dot\cup I_2, i \in I\}\!\}} \tag{15}$$

– *plus/minus axioms (for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$*):*

$$\frac{}{\{\!\{\{\lambda\} \to \{X_{J\cup\{i\}}\{\lambda\} \mid J \subseteq I^-\}, \{X_{\{1,\ldots,n\}}\{\lambda\}\} \to \{X_{I^-}\{\lambda\}\}, \{X(X_j\{\lambda\})\} \to \{X\}, \{\lambda\} \to \{X(X_k\{\lambda\})\} \mid i,j \in I^+, k \in I^-\}\!\}}\ \mathcal{C}$$

with condition $\mathcal{C} : \{1,\ldots,n\} = I^+ \dot\cup I^-$ *and*

$$\frac{}{\{\!\{\{X_{I^-}\{\lambda\}\} \to \{X_{J\cup\{i\}}\{\lambda\} \mid J \subseteq I^-\}, \{X_{K\cup\{\ell\}}\{\lambda\}\} \to \{X_K\{\lambda\}\}, \{X(X_j\{\lambda\})\} \to \{X\}, \{\lambda\} \to \{X(X_k\{\lambda\})\} \mid i,j \in I^+, k \in I^-\}\!\}}\ \mathcal{C}$$

$$\text{with condition } \mathcal{C} : \{1,\ldots,n\} = I^+ \dot\cup I^-, K \subseteq I^-, \ell \in I^+ \tag{16}$$

– *weakening rule:*

$$\frac{\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}}{\{\!\{\mathcal{Y}_j \to \mathcal{Z}_j \mid j \in J\}\!\}}\ I \subseteq J \tag{17}$$

– *left union rule:*

$$\frac{\{\!\{\mathcal{Y} \to \mathcal{Z}_i \mid i \in I\}\!\}}{\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}}\ \mathcal{Y} = \bigcup_{i\in I} \mathcal{Y}_i \tag{18}$$

– *shift rule:*

$$\frac{\{\!\{\mathcal{Y} \cup \mathcal{U}_1 \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}_1)\}\!\} \ldots \{\!\{\mathcal{Y} \cup \mathcal{U}_k \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}_k)\}\!\}}{\{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}}\ \mathcal{C}$$

$$\text{with condition } \mathcal{C} : \ \mathcal{P}(\mathcal{U}) = \{\mathcal{U}_1,\ldots,\mathcal{U}_k\} \tag{19}$$

Proof. The soundness of the weakening rule (17) is trivial. Therefore, we concentrate only on the other axioms and rules. For the union axiom (14) let $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n') = X(X_1\{X_1'\},\ldots,X_n\{X_n'\})$, $Y = X_I\{\lambda\}$, $Z_1 = X_J\{\lambda\}$ and $Z_2 = X(X(X_{i_1}\{X_{i_1}'\},\ldots,X_{i_k}\{X_{i_k}'\})$. Let $t_1, t_2 \in r$ with $\pi_Y^X(t_1) = \pi_Y^X(t_2)$ and $\pi_{Z_1}^X(t_1) \neq \pi_{Z_1}^X(t_2)$. Thus, one of t_1 or t_2 — without loss of generality let this be t_2 — must not contain elements of the form $(X_j : v_j)$ with $j \in J$. On the other hand, either r_1 and t_2 both contain elements of the form $(X_i : v_i)$ with $i \in I$ or both do not. As $I \subsetneq J$, it follows $\pi_{X(X_i\{\lambda\})}^X(t_1) = \pi_{X(X_i\{\lambda\})}^X(t_2) = \emptyset$ for all $i \in I$, which implies $\pi_{Z_2}^X(t_1) = \pi_{Z_2}^X(t_2)$.

oarieli@mta.ac.il

For the first partition axiom (15) let $t_1, t_2 \in r$ with $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_I\{\lambda\}}(t_2)$ and $\pi^X_{X(X_i\{\lambda\})}(t_1) \neq \pi^X_{X(X_i\{\lambda\})}(t_2)$ for all $i \in I$. Let $I_j \subseteq I$ be such that t_j contains an element of the form $(X_i : v_i)$ for all $i \in I_j$ $(j = 1, 2)$. Obviously, $I = I_1 \dot\cup I_2$ and $\pi^X_{X_{I'}\{\lambda\}}(t_1) = \pi^X_{X_{I'}\{\lambda\}}(t_2)$ for all $I' \subseteq I$ with $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$.

For the second partition axiom (15) assume $\pi^X_{X(X_i\{\lambda\})}(t_1) \neq \pi^X_{X(X_i\{\lambda\})}(t_2)$ for all $i \in I$ and $\pi^X_{X_{I'}\{\lambda\}}(t_1) = \pi^X_{X_{I'}\{\lambda\}}(t_2)$ for all $I' \subseteq I$ with $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$, in particular, $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_I\{\lambda\}}(t_2)$. Using the same construction of I_1 and I_2 again, we obtain $\pi^X_{X_{I_1}\{\lambda\}}(t_1) \neq \pi^X_{X_{I_1}\{\lambda\}}(t_2)$, which proves the claim.

For the first plus/minus axiom in (16) let t_1, t_2 satisfy $\pi^X_{X_j\{\lambda\}}(t_1) = \pi^X_{X_j\{\lambda\}}(t_2)$ and $\pi^X_{X_k\{\lambda\}}(t_1) \neq \pi^X_{X_k\{\lambda\}}(t_2)$ for all $j \in I^+$ and $k \in I^-$. Assume that for all $i \in I^+$ there is some $J \subseteq I^-$ with $\pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_2)$, i.e. one of these projections must be $\emptyset$. As we have $\pi^X_{X_i\{\lambda\}}(t_1) = \pi^X_{X_i\{\lambda\}}(t_2)$, these must both be $\emptyset$, which implies $\pi^X_{X_{I^+}\{\lambda\}}(t_j) = \emptyset$ for $j = 1, 2$. Now $\pi^X_{X_k\{\lambda\}}(t_1) \neq \pi^X_{X_k\{\lambda\}}(t_2)$ for all $k \in I^-$, so if $\pi^X_{X_{I^-}\{\lambda\}}(t_1) \neq \pi^X_{X_{I^-}\{\lambda\}}(t_2)$ holds, one of these projections must be $\emptyset$ again, which implies that one t_j is $\emptyset$, the other not empty. That is $\pi^X_{X_{\{1,\ldots,n\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{\{1,\ldots,n\}}\{\lambda\}}(t_2)$.

For the second plus/minus axiom in (16) let again t_1, t_2 satisfy $\pi^X_{X_j\{\lambda\}}(t_1) = \pi^X_{X_j\{\lambda\}}(t_2)$ and $\pi^X_{X_k\{\lambda\}}(t_1) \neq \pi^X_{X_k\{\lambda\}}(t_2)$ for all $j \in I^+$ and $k \in I^-$. Now assume $\pi^X_{X_{I^-}\{\lambda\}}(t_1) = \pi^X_{X_{I^-}\{\lambda\}}(t_2)$. Then, argueing in the same way as for the first partition axiom we obtain a partition $I^- = I_1 \dot\cup I_2$ with $\pi^X_{X_{I'}\{\lambda\}}(t_1) \neq \pi^X_{X_{I'}\{\lambda\}}(t_2)$, whenever $\emptyset \neq I' \subseteq I_1$ or $\emptyset \neq I' \subseteq I_2$, and $\pi^X_{X_{I'}\{\lambda\}}(t_1) = \pi^X_{X_{I'}\{\lambda\}}(t_2)$ for all I' with $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$. Now assume that for all $i \in I^+$ there is some $J \subseteq I^-$ with $\pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_2)$. Same as for the first plus/minus axiom we conclude $\pi^X_{X_{I^+}\{\lambda\}}(t_j) = \emptyset$ for $j = 1, 2$. Now, if $\pi^X_{X_K\{\lambda\}}(t_1) \neq \pi^X_{X_K\{\lambda\}}(t_2)$, i.e. $K \subseteq I_1$ or $K \subseteq I_2$, then we also have $\pi^X_{X_{K\cup\{\ell\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{K\cup\{\ell\}}\{\lambda\}}(t_2)$.

For the left union rule (18) assume $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$, i.e. there exist $t_1, t_2 \in r$ such that for all $i \in I$ we get $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}_i$ and $\pi^X_{Z_i}(t_1) \neq \pi^X_{Z_i}(t_2)$ for some $Z_i \in \mathcal{Z}_i$. In particular, $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$, hence $r \not\models \{\!\{\mathcal{Y} \to \mathcal{Z}_i \mid i \in I\}\!\}$.

For the shift rule (19) assume $r \not\models \{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}$, i.e. there exist $t_1, t_2 \in r$ such that $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ and $\pi^X_Z(t_1) \neq \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$. Take a maximal $\mathcal{U}' \subseteq \mathcal{U}$ such that $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}'$. If we had $r \models \{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\}$, we would have $\mathcal{U}' \subsetneq \mathcal{U}$, and there would exist some $V \in \mathcal{U} - \mathcal{U}'$ with $\pi^X_V(t_1) = \pi^X_V(t_2)$, which contradicts the maximality of $\mathcal{U}$. □

oarieli@mta.ac.il

Using the axioms and rules for wFDs we may also derive additional rules for FDs:

$$\frac{\{X(X_i\{\lambda\}), X\{\lambda\}\} \to \{X(X_i\{X_i'\})\}}{\{X(X_i\{\lambda\})\} \to \{X(X_i\{X_i'\})\}} \; X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$$

$$\frac{\{X(X_i\{X_i'\})\} \to \{X\{\lambda\}\}}{\{X(X_i\{\lambda\})\} \to \{X\{\lambda\}\}} \; X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$$

For the first rule we use the following derivation, in which the last step applies the shift rule with $\phi = \{\!\{\{X(X_i\{\lambda\})\} \to \{X(X_i\{X_i'\})\}, \{X(X_i\{\lambda\})\} \to \{X\{\lambda\}\}\}\!\}$ and $\mathcal{U} = \{X\{\lambda\}\}$:

$$\frac{\{\!\{\{X(X_i\{\lambda\}), X\{\lambda\}\} \to \{X(X_i\{X_i'\})\}\}\!\} \quad \overline{\phi}}{\{\!\{\{X(X_i\{\lambda\})\} \to \{X(X_i\{X_i'\})\}\}\!\}}$$

For the second rule we use the following derivation, in which we apply the shift rule (19) with $\mathcal{U} = \{X(X_i\{X_i'\})\}$ and the same ϕ:

$$\frac{\dfrac{\overline{\{X(X_i\{\lambda\}), X(X_i\{X_i'\})\} \to \{X(X_i\{X_i'\})\}} \quad \{X(X_i\{X_i'\})\} \to \{X\{\lambda\}\}}{\{\!\{\{X(X_i\{\lambda\}), X(X_i\{X_i'\})\} \to \{X\{\lambda\}\}\}\!\}} \quad \overline{\phi}}{\{\!\{\{X(X_i\{\lambda\})\} \to \{X\{\lambda\}\}\}\!\}}$$

4 SHL-Ideals

In this section we investigate ideals. Of particular interest will be ideals with additional closure properties, which we call "strong high-level ideals" or SHL-ideals for short. These ideals will appear naturally in the completeness proof in the next section. The main result of this section is Lemma 1.

In general, an *ideal* for a nested attribute X is a subset $\mathcal{G} \subseteq \mathcal{S}(X)$ with $\lambda \in \mathcal{G}$ and whenever $Y \in \mathcal{G}$, $Z \in \mathcal{S}(X)$ with $Y \geq Z$, then also $Z \in \mathcal{G}$. Let us now address the closure properties that will turn ideals into "higher-level" or "strong higher-level ideals".

Definition 10. Let $X \in \mathcal{N}$. An *SHL-ideal* on $\mathcal{S}(X)$ is a subset $\mathcal{F} \subseteq \mathcal{S}(X)$ with the following properties:

1. $\lambda \in \mathcal{F}$;
2. if $Y \in \mathcal{F}$ and $Z \in \mathcal{S}(X)$ with $Y \geq Z$, then $Z \in \mathcal{F}$;
3. if $Y, Z \in \mathcal{F}$ are semi-disjoint, then $Y \sqcup Z \in \mathcal{F}$;
4. a) if $X_I\{\lambda\} \in \mathcal{F}$ and $X_J\{\lambda\} \notin \mathcal{F}$ for $I \subsetneq J$, then $X(X_{i_1}\{X_{i_1}'\}, \ldots, X_{i_k}\{X_{i_k}'\}) \in \mathcal{F}$ for $I = \{i_1, \ldots, i_k\}$;
   b) if $X_I\{\lambda\} \in \mathcal{F}$ and $X(X_i\{\lambda\}) \in \mathcal{S}(X) - \mathcal{F}$ for all $i \in I$, then there is a partition $I = I_1 \dot\cup I_2$ with $X_{I_1}\{\lambda\} \notin \mathcal{F}$, $X_{I_2}\{\lambda\} \notin \mathcal{F}$ and $X_{I'}\{\lambda\} \in \mathcal{F}$ for all $I' \subseteq I$ with $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$;
   c) if $X_{\{1,\ldots,n\}}\{\lambda\} \in \mathcal{F}$ and $X_{I^-}\{\lambda\} \notin \mathcal{F}$ (for $I^- = \{i \in \{1, \ldots, n\} \mid X(X_i\{\lambda\}) \notin \mathcal{F}\}$), there there exists some $i \in I^+ = \{i \in \{1, \ldots, n\} \mid X(X_i\{\lambda\}) \in \mathcal{F}\}$ such that for all $J \subseteq I^-$ $X_{J \cup \{i\}}\{\lambda\} \in \mathcal{F}$ holds;

oarieli@mta.ac.il

d) if $X_{I^-}\{\lambda\} \in \mathcal{F}$ and for all $i \in I^+ = \{i \in \{1,\ldots,n\} \mid X(X_i\{\lambda\}) \in \mathcal{F}\}$ there is some $J \subseteq I^-$ with $X_{J\cup\{i\}}\{\lambda\} \notin \mathcal{F}$, then for all $\ell \in I^+$ and all $K \subseteq I^-$ with $X_K\{\lambda\} \notin \mathcal{F}$ we also have $X_{K\cup\{\ell\}}\{\lambda\} \notin \mathcal{F}$;

5. if $X_I\{\lambda\} \in \mathcal{F}$ and $X_J\{\lambda\} \in \mathcal{F}$ with $I \cap J = \emptyset$, then $X_{I\cup J}\{\lambda\} \in \mathcal{F}$;
6. a) if $X = X(X'_1,\ldots,X'_n)$, then $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X'_i) \mid X(\lambda,\ldots,Y_i,\ldots,\lambda) \in \mathcal{F}\}$ is an SHL-ideal;
   b) If $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ and $\mathcal{F} \neq \{\lambda\}$, then the set $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X'_i) \mid X_1(\lambda) \oplus \cdots \oplus X_i(Y_i) \oplus \cdots \oplus X_n(\lambda) \in \mathcal{F}\}$ is an SHL-ideal;
   c) if $X = X\{X'\}$ and $\mathcal{F} \neq \{\lambda\}$, then $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X\{Y\} \in \mathcal{F}\}$ is a semi-SHL-ideal.

An ideal satisfying properties 1 – 3 will be called *HL-ideal*. An ideal is called a *semi-SHL-ideal* iff it satisfies properties 1, 2, 4 and a modification of 6, in which we only require a semi-SHL-ideal instead of an SHL-ideal.

A semi-SHL-ideal $\mathcal{G}$ can be written as a union of SHL-ideals, say $\mathcal{G} = \mathcal{H}_1 \cup \cdots \cup \mathcal{H}_k$. If there is no inclusion among these SHL-ideals, we say that $\{\mathcal{H}_1,\ldots,\mathcal{H}_k\}$ *covers* $\mathcal{G}$.

The next lemma is the main result of this section. Its proof is very technical and lengthy, and is omitted here.

Lemma 1 (Central Lemma). *Let X be a nested attribute such that the union-constructor appears in X only inside a set-constructor. If $\mathcal{F}$ is a SHL-ideal on $\mathcal{S}(X)$, then there exist tuples $t_1, t_2 \in dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y \in \mathcal{F}$. Furthermore, if $\mathcal{G}$ is a semi-SHL-ideal on $\mathcal{S}(X)$, then there are finite sets $S_1, S_2 \subseteq dom(X)$ with $\{\pi^X_Y(\tau) \mid \tau \in S_1\} = \{\pi^X_Y(\tau) \mid \tau \in S_2\}$ iff $Y \in \mathcal{G}$.*

5 The Completeness of the Derivation Rules

In this section we want to show that the axioms and rules from Section 3 are also complete. This gives our main result.

Before we come to the proof let us make a little observation on the union-constructor. If $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ the each instance r of X can be partioned into r_i $(i = 1,\ldots,n)$, where r_i contains exactly the X_i-labelled elements of r. Then r satisfies a FD $\varphi \equiv \mathcal{Y} \to \mathcal{Z}$ iff each r_i satifies the i'th projection φ_i of φ, which results by replacing all subattributes $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ in $\mathcal{Y}$ or $\mathcal{Z}$ by $X_i(Y_i)$. Similarly, we see $\varphi \in \Sigma^+$ iff $\varphi_i \in \Sigma^+_i$ for all $i = 1,\ldots,n$.

Theorem 3. *The axioms and rules in Theorems 1 and 2 are complete for the implication of wFDs.*

Proof. Let Σ be a set of wFDs on $\mathcal{S}(X)$ and assume $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^+$. Due to the union rule (11) we must have $\{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ for some selected $Z_i \in \mathcal{Z}_i$. Furthermore, due to the left union rule (18) we get $\{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ with $\mathcal{Y} = \bigcup_{i\in I} \mathcal{Y}_i$.

Let $\mathcal{Z} = \{Z \mid Z \geq Z_i \text{ for some } i \in I\}$ and $\mathcal{U} = \mathcal{S}(X) - \mathcal{Y} - \mathcal{Z}$. Due to the reflexivity axiom (4) we obviously have $Z_i \notin \mathcal{Y}$, and then $\mathcal{Y} \sqcap \mathcal{Z} = \emptyset$ due to the

oarieli@mta.ac.il

subattribute axiom (2). Due to the shift rule (19) there must exist some $\mathcal{U}' \subseteq \mathcal{U}$ with $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$. Otherwise we could derive $\{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}$, and thus $\{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \in \Sigma^+$ contradicting our assumption.

Take $\mathcal{U}'$ maximal with the given property. We show that $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$ is a SHL-ideal.

1. Assume $\mathcal{F} = \emptyset$. This implies $\mathcal{Z} \cup \mathcal{U} = \mathcal{S}(X)$ and thus $\{\!\{\emptyset \to\!\!\to \{Z\} \mid Z \in \mathcal{S}(X)\}\!\} \notin \Sigma^+$. This wFD, however, can be derived from $\{\!\{\emptyset\{\lambda\}\}\!\} \in \Sigma^+$ (due to the λ axiom (1)) using the weakening rule (17). Thus, $\mathcal{F}$ is not empty.
2. Now let $Y \in \mathcal{F}$ and $Y \geq Y'$ Assume $Y' \notin \mathcal{F}$. So $Y' \in \mathcal{U}$, otherwise we get $Y' \in \mathcal{Y} \cup \mathcal{Z}$, which implies $Y' \geq Z_i$ for some $i \in I$ and furtheron $Y \geq Y' \geq Z_i$, which gives the contradiction $Y \in \mathcal{Z}$.
   Now take $\mathcal{U}'' = \mathcal{U}' \cup \{Y'\}$. The subattribute axiom (2) together with the extension and transitivity rules (5) and (6) implies $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{Y} \cup \mathcal{U}'' \in \Sigma^+$. As $\mathcal{U}'$ was chosen maximal, we also have $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$. Using the transitivity rule (6) again, this gives $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$. Then the weakening rule (17) leads to the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.
3. Let $Y_1, Y_2 \in \mathcal{F}$ be semi-disjoint. Assume $Y = Y_1 \sqcup Y_2 \notin \mathcal{F}$. If $Y \in \mathcal{U}$, we take $\mathcal{U}'' = \mathcal{U}' \cup \{Y\}$. Due to the maximality of $\mathcal{U}'$ we get $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$, thus by the weakening rule (17) also $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.
   On the other hand, the join axiom (3) implies $\{Y_1, Y_2\} \to \{Y\} \in \Sigma^+$. Using the reflexivity axiom (4), the extension rule (5) and the transitivity rule (6) we obtain $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{Y} \cup \mathcal{U}'' \in \Sigma^+$, from which we get the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$ by another application of the transitivity rule.
   If $Y \notin \mathcal{U}$, we get $Y \in \mathcal{Z}$, thus $\{Y\}$ is among the right hand sides in $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$. However, the join rule (13) together with the reflexivity axiom and the transitivity rule imply $\mathcal{Y} \cup \mathcal{U}' \to \{Y\} \in \Sigma^+$, hence the weakening rule leads to the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.
4. Assume $X_I\{\lambda\} \in \mathcal{F}$, but $(X_J\{\lambda\} \notin \mathcal{F}$ for $\{i_1, \ldots, i_k\} = I \subsetneq J$. As $\mathcal{S}(X)$ is partitioned into $\mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$ and $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$, we must have $X_J\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. From the union axiom (14), the transitivity rule and $X(X_I\{\lambda\}) \in \mathcal{F}$ we conclude $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \{X_J\{\lambda\}, X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})\}\}\!\} \in \Sigma^+$. Due to the weakening rule (17) it follows $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{W}\}\!\} \in \Sigma^+$ for all $\mathcal{W} \subseteq \mathcal{S}(X)$ with $X_J\{\lambda\}, X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \in \mathcal{W}$. According to the definition of $\mathcal{U}'$ we must have either $X_J\{\lambda\} \notin \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$ or $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \notin \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, which implies $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \in \mathcal{F}$.
   The same argument applies for $X\{X'(X_I\{\lambda\})\} \in \mathcal{F}$, but $X\{X'_J\{\lambda\}\} \notin \mathcal{F}$ for $\{i_1, \ldots, i_k\} = I \subsetneq J$. Just use the set lifting rule (8) in addition. This implies $X\{X'(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})\} \in \mathcal{F}$. This shows that the embedded ideal $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X[Y] \in \mathcal{F}\}$ satisfies this property, too.

oarieli@mta.ac.il

Furthermore, the same argument applies for $X(\lambda,\ldots,X_{i_I}\{\lambda\},\ldots,\lambda) \in \mathcal{F}$ and $X(\lambda,\ldots,X_{i_J}\{\lambda\},\ldots,\lambda) \notin \mathcal{F}$ for $\{i_1,\ldots,i_k\} = I \subsetneq J$. Just use the record lifting rule (9) in addition.
This implies $X(\lambda,\ldots,X_i(X_{i_1}\{X'_{i_1}\},\ldots,X_{i_k}\{X'_{i_k}\}),\ldots,\lambda) \in \mathcal{F}$, and shows that the embedded ideal $\mathcal{G}_i = \{Y_i \in \mathcal{S}(X_i) \mid X(\lambda,\ldots,Y_i,\ldots,\lambda) \in \mathcal{F}\}$ satisfies this property, too. Analogously, using the lifting rule for unions we can show this property for the embedded ideals induced by the union-constructor.

5. Assume $X_I\{\lambda\} \in \mathcal{F}$, but $X(X_i\{\lambda\}) \notin \mathcal{F}$ for all $i \in I$. In particular $X(X_i\{\lambda\}) \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. Using the first partition axiom (15), the transitivity rule and $X_I\{\lambda\} \in \mathcal{F}$ we conclude $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{X_{I'_1 \cup I'_2}\{\lambda\}, \mathcal{Y} \cup \mathcal{U}' \to \{X(X_i\{\lambda\})\} \mid \emptyset \neq I'_1 \subseteq I_1, \emptyset \neq I'_2 \subseteq I_2\} \mid I = I_1 \dot\cup I_2, i \in I\}\!\} \in \Sigma^+$.
If for all partitions $I = I_1 \dot\cup I_2$ we had at least one $X_{I'_1 \cup I'_2}\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, we can apply the reflexivity axiom, the transitivity rule and the weakening rule to derive $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$ contradicting the assumption on $\mathcal{U}'$. Therefore, there is a partition $I = I_1 \dot\cup I_2$ with $\{X_{I'_1 \cup I'_2}\{\lambda\} \mid \emptyset \neq I'_1 \subseteq I_1, \emptyset \neq I'_2 \subseteq I_2\} \subseteq \mathcal{F}$. If we had $X_{I_1}\{\lambda\} \in \mathcal{F}$ for all such partitions, we would get $\{X_{I'}\{\lambda\} \mid I' \cap I_1 \neq \emptyset \neq I' \cap I_2\} \cup \{X_{I_1}\{\lambda\}\} \subseteq \mathcal{F}$, thus using the reflexivity axiom, the transitivity rule, the second partition axiom, the fact $X(X_i\{\lambda\}) \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$ for all $i \in I$ holds, and the weakening rule, we obtain again the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$. Thus, $X_{I_1}\{\lambda\}, X_{I_2}\{\lambda\} \notin \mathcal{F}$.
Using again the lifting rules we obtain the corresponding property for embedded ideals.
6. Assume $X_{\{1,\ldots,n\}}\{\lambda\} \in \mathcal{F}$, $X_{I^-}\{\lambda\} \notin \mathcal{F}$ and for all $i \in I^+$ there is some $J \subseteq I^-$ with $X_{J \cup \{i\}}\{\lambda\} \notin \mathcal{F}$. Let this J be denoted as J_i. Taking the first plus/minus axiom (16), the left hand side of the FDs are always in $\mathcal{F}$. Therefore, using the reflexivity axiom and the transitivity rule we derive $\{\!\{\mathcal{F} \to \{X_{J_i}\{\lambda\}\}, \mathcal{F} \to \{X\}, \mathcal{F} \to \{X_j\{\lambda\}\}, \mathcal{F} \to \{X_{I^-}\{\lambda\}\} \mid i \in I^+, j \in I^-\}\!\} \in \Sigma^+$. Now the right hand sides of the FDs are all not in $\mathcal{F}$, so the weakening rule implies $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$ contradicting the construction of $\mathcal{F}$, according to which $\mathcal{S}(X) - \mathcal{F} = \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. and $\{\!\{\mathcal{F} \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$.
7. Assume $X_{I^-}\{\lambda\} \in \mathcal{F}$ and for all $i \in I^+$ there is some $J \subseteq I^-$ with $X_{J \cup \{i\}}\{\lambda\} \notin \mathcal{F}$ – let this be denoted as J_i – and further assume there is some $\ell \in I^+$ and some $K \subseteq I^-$ with $X_K\{\lambda\} \notin \mathcal{F}$, but $X_{K \cup \{\ell\}}\{\lambda\} \in \mathcal{F}$. Taking the second plus/minus axiom for this K and ℓ, all left hand side of the FDs are always in $\mathcal{F}$. Therefore, using the reflexivity axiom and the transitivity rule we derive $\{\!\{\mathcal{F} \to \{X_{J_i}\{\lambda\}\}, \mathcal{F} \to \{X\}, \mathcal{F} \to \{X_j\{\lambda\}\}, \mathcal{F} \to \{X_{I^-}\{\lambda\}\} \mid i \in I^+, j \in I^-\}\!\} \in \Sigma^+$. Now the right hand sides of the FDs are all not in $\mathcal{F}$, so the weakening rule implies $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$ contradicting $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \notin \Sigma^+$.
8. Assume $X_I\{\lambda\}, X_J\{\lambda\} \in \mathcal{F}$ with $I \cap J = \emptyset$. If $X_{I \cup J}\{\lambda\} \notin \mathcal{F}$, then $X_{I \cup J}\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. As before we can derive $\mathcal{Y} \cup \mathcal{U}' \to \{X_{I \cup J}\{\lambda\}\} \in \Sigma^+$ using the reflexivity axiom, the second multiset axiom and the transitivity rule. Then the application of the weakening rule leads to $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$ contradicting the assumption on $\mathcal{U}'$.

oarieli@mta.ac.il

Due to the restructuring rules in Definition 4 we may assume that only the union-constructor appears in X only inside a set-, list- or multiset-constructor or as the outermost constructor.

Let us first assume that the outermost constructor is not the union-constructor. Then we can apply the Central Lemma 1, which gives us $r = \{t_1, t_2\} \subseteq dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y \in \mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$. In particular, $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $i \in I$ and $Y \in \mathcal{Y}_i$, and $\pi^X_{Z_i}(t_1) \neq \pi^X_{Z_i}(t_2)$ for all $i \in I$. That is, $r \not\models \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\}$. From the soundness of the fragmentation rule (12) we conclude $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

Now assume that the outermost constructor of X is the union-constructor, say $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$. We know that $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$ is a SHL-ideal on $\mathcal{S}(X)$. If $\mathcal{F} = \{\lambda\}$, then take $t_1 = (X_1 : t'_1)$ and $t_2 = (X_2 : t'_2)$ with arbitrary $t'_j \in dom(X'_j)$. Then $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y = \lambda$. As before this implies $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with $r = \{t_1, t_2\}$.

For $\mathcal{F} \neq \{\lambda\}$ take the embedded SHL-ideal $\mathcal{F}_i$ on $\mathcal{S}(X'_i)$. Using the Central Lemma 1 we find $t_{i1}, t_{i2} \in dom(X'_i)$ with $\pi^{X'_i}_{Y_i}(t_{i1}) = \pi^{X'_i}_{Y_i}(t_{i2})$ iff $Y_i \in \mathcal{F}_i$.

As we have $\{\!\{\mathcal{F} \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$, we must also have $\{\!\{\mathcal{F}_j \to \{Z\} \mid Z \in (\mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'))_j\}\!\} \notin \Sigma^+_j$ for at least one j. In particular, for $Z_i = X_1(Z'_{i1}) \oplus \cdots \oplus X_n(Z'_{in})$ we find some j such that $Z'_{ij} \notin \mathcal{F}_j$ for all $i \in I$.

Now take $r = \{(X_j : t_{j1}), (X_j : t_{j2})\}$. Then for all $i \in I$ and all $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \in \mathcal{Y}_i \subseteq \mathcal{F}$ we have $Y_j \in \mathcal{F}_j$, and we obtain

$$\pi^X_Y((X_j : t_{j1})) = (X_j : \pi^{X'_j}_{Y_j}(t_{j1})) = (X_j : \pi^{X'_j}_{Y_j}(t_{j2})) = \pi^X_Y((X_j : t_{j2}))\,.$$

On the other hand, $Z'_{ij} \notin \mathcal{F}_j$ implies

$$\pi^X_{Z_i}((X_j : t_{j1})) = (X_j : \pi^{X'_j}_{Z'_{ij}}(t_{j1})) \neq (X_j : \pi^{X'_j}_{Z'_{ij}}(t_{j2})) = \pi^X_{Z_i}((X_j : t_{j2}))$$

for all $i \in I$. That is $r \not\models \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\}$, and hence $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ by the soundness of the fragmentation rule (12).

We will now show $r \models \Sigma$ in both cases. This implies $r \models \Sigma^*$, and thus $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^*$, which completes the proof.

First assume again that the outermost constructor is not the union-constructor. Let $\{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\} \in \Sigma$.

1. If $\mathcal{V}_j \not\subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$, we get $\pi^X_V(t_1) \neq \pi^X_V(t_2)$ for some $V \in \mathcal{V}_j$. Thus $r \models \mathcal{V}_j \to \mathcal{W}_j$ and due to the soundness of the weakening rule also $r \models \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$.
2. If $\mathcal{V}_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for all $j \in J$, we get $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{V}_j \in \Sigma^+$ from the reflexivity axiom, $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \mathcal{W}_j \mid j \in J\}\!\} \in \Sigma^+$ from the transitivity rule, and $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{W_j\} \mid j \in J\}\!\} \in \Sigma^+$ for any choices $W_j \in \mathcal{W}_j$ from the fragmentation rule. Assume we could select $W_j \in \mathcal{W}_j - \mathcal{Y} - \mathcal{U}'$ for all $j \in J$. Then the weakening rule implies $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{W\} \mid W \in \mathcal{S}(X) - \mathcal{Y} - \mathcal{U}'\}\!\} \in \Sigma^+$. However, $\mathcal{S}(X) - \mathcal{Y} - \mathcal{U}' = \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, so we get a contradiction to the choice of $\mathcal{U}'$.
   Therefore, we must have $\mathcal{W}_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$. By construction of r we get $\pi^X_W(t_1) = \pi^X_W(t_2)$ for all $W \in \mathcal{W}_j$, thus $r \models \mathcal{V}_j \to \mathcal{W}_j$. This implies $r \models \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$ due to the soundness of the weakening rule.

oarieli@mta.ac.il

If the outermost constructor is the union-constructor we just have to show $r \models \Sigma_j$. The proof is analogous to the case before. □

This main theorem shows the axiomatisation of wFDs. If Σ is a set of "ordinary" FDs, we can apply the axioms and rules to Σ and then the FDs in Σ^+ will be the implied FDs. Of course, we would like to have an axiomatisation for FDs that avoids such a detour via the wFDs.

6 Conclusion

In this paper we investigated functional dependencies and weak functional dependencies (i.e. disjunctions of functional dependencies) in the presence of a record constructor, a constructor for the null value "not exists", a finite set constructor, and a disjoint union constructor. We achieved a finite axiomatisation for weak functional dependencies.

The main technical tool for the completeness proof was a central lemma on SHL-ideals, i.e. ideals with certain additional closure properties as they arise from closures of sets of subattributes with respect to a given set of functional dependencies. Roughly speaking the central lemma guarantees the existence of values that coincide exactly on the SHL-ideal. The proof of the central lemma is quite elegant, as long as the union-constructor is absent. However, in the presence of the union-constructor, the proof becomes a bit awkward. Fortunately, the presence of restructuring rules for the union-constructor allows us to assume that the union constructor only appears as the outermost constructor or inside a set-constructor.

The next steps are to extend the theory in several directions covering multivalued dependencies, error-robustness, null values "unknown", and references. We strongly conjecture that the results on SHL-ideals will turn out again to be a valuable proof tool.

Furthermore, we would like to extend the theory also to multisets and lists, which add further restructuring rules. At the time of completing this paper the corresponding proof of the Cover Lemma including these two constructors was still flawed. Finally, we may think of projecting lists onto multisets — just forget the order of the elements — and multisets onto sets — forget the multiplicities. This would further extend the lattices of subattributes.

References

1. Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, 2000.
2. Abiteboul, S., and Hull, R. Restructuring hierarchical database objects. *Theoretical Computer Science* (1988).
3. Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995.
4. Arenas, M., and Libkin, L. A normal form for XML documents. In *PODS 2002* (2002), ACM.

oarieli@mta.ac.il

5. Armstrong, W. W. Dependency structures of database relationships. *Information Processing* (1974), 580–583.
6. Batini, C., Ceri, S., and Navathe, S. B. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin Cummings, 1992.
7. Buneman, P., Davidson, S., Fan, W., Hara, C., and Tan, W. Keys for XML. In *Tenth WWW Conference* (2001), IEEE.
8. Chen, P. P. The entity-relationship model: Towards a unified view of data. *ACM Transactions Database Systems 1* (1976), 9–36.
9. Chen, P. P. English sentence structure and entity-relationship diagrams. *Information Science 29* (1983), 127–149.
10. Fan, W., and Libkin, L. On XML integrity constraints in the presence of DTDs. In *PODS 2001* (2001), ACM.
11. Fan, W., and Siméon, J. Integrity constraints for XML. In *PODS 2000* (2000), ACM.
12. Hartmann, S. Decomposing relationship types by pivoting and schema equivalence. *Data & Knowledge Engineering 39* (2001), 75–99.
13. Hartmann, S., and Link, S. Reasoning about functional dependencies in an abstract data model. *Electronic Notes in Theoretical Computer Science 84* (2003).
14. Hartmann, S., Link, S., and Schewe, K.-D. Capturing functional dependencies in the higher-order entity relationship model. Tech. Rep. 5/2003, Massey University, Department of Information Systems, 2003.
15. Hartmann, S., Link, S., and Schewe, K.-D. Generalizing Boyce-Codd normal form to conceptual databases. In *Pre-Proceedings of the 13th European-Japanese Conference on Information Modeling and Knowledge Bases* (2003), pp. 93–110.
16. Hull, R., and King, R. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys 19*, 3 (1987).
17. Mok, W. Y., Ng, Y. K., and Embley, D. W. A normal form for precisely charachterizing redundancy in nested relations. *Transactions on Database Systems 21* (1996), 77–106.
18. Özsoyoglu, Z. M., and Yuan, L. Y. A new normal form for nested relations. *Transactions on Database Systems 12* (1987), 111–136.
19. Paredaens, J., De Bra, P., Gyssens, M., and Van Gucht, D. *The Structure of the Relational Database Model*. Springer-Verlag, 1989.
20. Schewe, K.-D., and Thalheim, B. Fundamental concepts of object oriented databases. *Acta Cybernetica 11*, 4 (1993), 49–85.
21. Tari, Z., Stokes, J., and Spaccapietra, S. Object normal forms and dependency constraints for object-oriented schemata. *ACM ToDS 22* (1997), 513–569.
22. Thalheim, B. *Dependencies in Relational Databases*. Teubner-Verlag, 1991.
23. Thalheim, B. Foundations of entity-relationship modeling. *Annals of Mathematics and Artificial Intelligence 6* (1992), 197–256.
24. Thalheim, B. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag, 2000.
25. Tjoa, A. M., and Berger, L. Transformation of requirement specifications expressed in natural language into an eer model. In *Entity-Relationship Approach* (1993), vol. 823, Springer Lecture Notes Series.
26. Vincent, M. *The semantic justification for normal forms in relational database design*. PhD thesis, Monash University, Melbourne, Australia, 1994.

oarieli@mta.ac.il

Reasoning about Functional and Multi-valued Dependencies in the Presence of Lists

Sven Hartmann, Sebastian Link, and Klaus-Dieter Schewe

Information Science Research Centre, Massey University
Private Bag 11222, Palmerston North, New Zealand
phone: +64 6 350 5799 extn 2717
fax: +64 6 350 5725
{s.hartmann,s.link,k.d.schewe}@massey.ac.nz

Abstract. Nested lists are used as a data structure whenever order matters. List types are therefore supported by many advanced data models such as genomic sequence, deductive and object-oriented data models including XML.
It is studied what impact the presence of the finite list type has on the two most important classes of relational dependencies. A finite axiomatisation of functional and multi-valued dependencies in databases supporting base, record and finite list types is presented. In order to capture different data models at a time, an abstract algebraic approach based on nested attributes and subtyping is taken. This algebraic framework together with a new inference rule allowing to derive non-trivial functional dependencies from multi-valued dependencies make the generalisation of the well-known theory from the relational data model natural.

1 Introduction

In designing databases the semantics of the application domain has to be captured as completely as possible. As this cannot be expressed solely by structures, we have to use dependencies, i.e., sentences in a logic suitable for the data model used. Database theory has to investigate the implications arising from the presence of dependencies. This means to describe semantically desirable properties of "well-designed" databases, e.g., the absence of redundancy, to characterise (if possible) them syntactically by in-depth investigation of the dependencies and to develop algorithms to transform schemata into normal forms, which guarantee the desirable properties to be satisfied.

In the relational data model (RDM, [2,34]) a lot of research has been done on dependency theory and normal forms. Starting with the seminal work by Codd [18] normal forms such as third normal form (3NF), Boyce-Codd normal form (BCNF, [10,19]) and fourth normal form (4NF, [20,21,22]) have been introduced to characterise the absence of redundancy and update anomalies in the presence of functional and multi-valued dependencies (FDs, MVDs), though a theoretically convincing justification for these normal forms was given only 20 years later [42]. Roughly speaking, a functional dependency $X \to Y$ requires

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 134–154, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

that whenever two tuples of a relation coincide on X, they must also coincide on Y. A multi-valued dependency $X \twoheadrightarrow Y$ requires that whenever two tuples of a relation coincide on X, their values on Y must be mutually exchangeable and thus generate additional tuples.

Various other classes of dependencies for the RDM have been introduced (see [39] for an overview) and large parts of database theory deals with the finite axiomatisation of these dependencies and the finite implication problem for them, i.e., to decide that a dependency φ is implied by a set of dependencies Σ, where implication refers to the fact that all finite models of Σ are also models of φ. Armstrong [4] was the first to give a finite axiomatisation for FDs, and Beeri and others gave a finite axiomatisation for FDs and MVDs [9] and developed various versions of efficient decision algorithms [6,7,8].

During the last couple of decades, many new and different data models have been introduced. First, so called semantic data models have been developed [16, 28], which were originally just meant to be used as design aids, as application semantics was assumed to be easier captured by these models [5,17,41]. Later on some of these models, especially the nested relational model [34], object oriented models [36] and object-relational models, the gist of which are captured by the higher-order Entity-Relationship model (HERM, [40]) have become interesting as data models in their own right and some dependency and normalisation theory has been carried over to these advanced data models [24,26,27,31,33,34,38]. The work in [26] provides a finite axiomatisation of FDs in the presence of sets. The expressiveness, however, deviates from those in previous works on FDs in the nested relational model [31,33]. Therefore, a new normal form is proposed in [27] and it is semantically justified by formally proving the equivalence to the absence of redundancies and the sufficiency for the absence of any update anomalies. Most recently, the major research interest is on the model of semi-structured data and XML [1], which may also be regarded as some kind of object oriented model. The work in [3] considers FDs arising from a relational representation of XML documents. There are, however, different concepts of FDs in the context of XML, each resulting in a different expressiveness. See [25] for a detailed discussion. The only paper that studies MVDs in advanced data models is [43]. Their approach is similar to the one in [3], and no axiomatisation results are provided. The authors are not aware of any work in the literature that specifically deals with lists and the class of FDs and MVDs, nevermind an axiomatisation.

One key problem is to develop dependency theories (or preferably a unified theory) for the most relevant advanced data models. These are probably the HERM as a nested model with various bulk type constructors, good theoretical foundations and proven practical relevance [40], the object oriented model [36], the semi-structured data model and XML [1], which add unions and most importantly references, the expansion of which leads to rational tree structures. The development of such a dependency theory will have a significant impact on understanding application semantics and laying the grounds for a logically founded theory of well-designed databases. Biskup [13,14] lists in particular two challenges for database design theory: finding a unifying framework and extend-

oarieli@mta.ac.il

ing achievements to deal with advanced database features such as complex object types.

In order to pursue a unifying framework and capture several data models at a time our work is based on an abstract approach in the context of types for nested attributes and subtyping. In the present paper we consider the perhaps most common data type: finite lists. There are several reasons for this choice. The need for lists arises from applications that store ordered relations, time-series data, meteorological and astronomical data streams, runs of experimental data, multidimenstional arrays, textual information, voices, sound, images, video, etc. Lists have been subject to studies in the deductive and temporal database community for some time [35,32]. The list type also naturally appears in object-oriented databases [36,23] and is in particular important for XML [1, 44]. Recently, bioinformatics has become a very important field of research. Of course, lists occur naturally in genomic sequence databases [37,15].

The paper is organised as follows. The algebraic framework is introduced in Section 2. It is demonstrated that the set of subattributes for some fixed nested attribute carries the structure of a Brouwerian Algebra (co-Heyting Algebra). This generalises the framework of a Boolean Algebra from the RDM. We show in Section 3 how to obtain a sound and complete set of inference rules for the implication of FDs in the presence of lists. In Section 4 we add MVDs and generalise the well-known result that MVDs are satisfied by some relation exactly when this relation can be decomposed in two of its projections without loosing or adding information. The main result is a finite axiomatisation for the class of FDs and MVDs. The inference rules are natural generalisations of their counterparts from relational databases. It turns out, however, that an additional rule allowing the derivation of non-trivial FDs from MVDs is needed (which is impossible in the RDM). We briefly comment on future research in Section 5.

2 The Algebra of Nested Attributes

This section introduces a data model based on the nesting of attributes and subtyping. It may be used to provide a unifying framework for the study of complex object types such as records, lists, sets, multisets, unions and references. This article, however, focuses on records and lists.

2.1 Nested Attributes

We start with the definition of flat attributes and values for them.

Definition 2.1. A *universe* is a finite set $\mathcal{U}$ together with domains (, i.e., sets of values) $dom(A)$ for all $A \in \mathcal{U}$. The elements of $\mathcal{U}$ are called *flat attributes*. □

For the relational data model a universe was sufficient. That is, a relation schema is defined by a finite and non-empty subset $\mathcal{R} \subseteq \mathcal{U}$. For higher-order data models, however, nested attributes are needed. In the following definition we use a set $\mathcal{L}$ of labels, and assume that the symbol λ is neither a flat attribute nor a label, i.e., $\lambda \notin \mathcal{U} \cup \mathcal{L}$. Moreover, flat attributes are not labels and vice versa, i.e., $\mathcal{U} \cap \mathcal{L} = \emptyset$.

oarieli@mta.ac.il

Definition 2.2. Let $\mathcal{U}$ be a universe and $\mathcal{L}$ a set of labels. The set $\mathcal{NA} = \mathcal{NA}(\mathcal{U}, \mathcal{L})$ of *nested attributes over* $\mathcal{U}$ *and* $\mathcal{L}$ is the smallest set satisfying the following conditions:

- $\lambda \in \mathcal{NA}$,
- $\mathcal{U} \subseteq \mathcal{NA}$,
- for $L \in \mathcal{L}$ and $N_1, \ldots, N_k \in \mathcal{NA}$ with $k \geq 1$ we have $L(N_1, \ldots, N_k) \in \mathcal{NA}$,
- for $L \in \mathcal{L}$ and $N \in \mathcal{NA}$ we have $L[N] \in \mathcal{NA}$.

We call λ *null attribute*, $L(N_1, \ldots, N_k)$ *record-valued attribute* and $L[N]$ *list-valued attribute*. □

We can now extend the mapping *dom* from flat attributes to nested attributes, i.e., we define a set $dom(N)$ of values for every nested attribute $N \in \mathcal{NA}$.

Definition 2.3. For a nested attribute $N \in \mathcal{NA}$ we define the *domain* $dom(N)$ as follows:

- $dom(\lambda) = \{ok\}$,
- $dom(L(N_1, \ldots, N_k)) = \{(v_1, \ldots, v_k) \mid v_i \in dom(N_i) \text{ for } i = 1, \ldots, k\}$, i.e., the set of all k-tuples $(v_1, \ldots, v_k)$ with $v_i \in dom(N_i)$ for all $i = 1, \ldots, k$, and
- $dom(L[N]) = \{[v_1, \ldots, v_n] \mid v_i \in dom(N) \text{ for } i = 1, \ldots, n\}$, i.e., the set of all finite lists with elements in $dom(N)$. □

The empty list is denoted by [ ]. Note that the relational data model is completely covered by the presence of tuple-valued attributes only. Instead of relation schemata R we will now consider a nested attribute N, assuming that a universe $\mathcal{U}$ and a set $\mathcal{L}$ of labels are fixed. An R-relation r is then replaced by some finite set $r \subseteq dom(N)$.

2.2 Subattributes

Dependency theory in the relational data model is based on the powerset $\mathcal{P}(R)$ for a relation schema R. In fact, $\mathcal{P}(R)$ is a powerset algebra with partial order $\subseteq$, set union $\cup$, set intersection $\cap$ and set difference $-$. We will generalise these operations for nested attributes starting with a partial order $\leq$.

Definition 2.4. The *subattribute relation* $\leq$ on the set of nested attributes $\mathcal{NA}$ over $\mathcal{U}$ and $\mathcal{L}$ is defined by the following rules, and the following rules only:

- $N \leq N$ for all nested attributes $N \in \mathcal{NA}$,
- $\lambda \leq A$ for all flat attributes $A \in \mathcal{U}$,
- $\lambda \leq N$ for all list-valued attributes $N \in \mathcal{NA}$,
- $L(N_1, \ldots, N_k) \leq L(M_1, \ldots, M_k)$ whenever $N_i \leq M_i$ for all $i = 1, \ldots, k$,
- $L[N] \leq L[M]$ whenever $N \leq M$.

For $N, M \in \mathcal{NA}$ we say that M is a *subattribute* of N if and only if $M \leq N$ holds. We write $M \not\leq N$ if and only if M is not a subattribute of N. □

oarieli@mta.ac.il

The subattribute relation $\leq$ on nested attributes is reflexive, anti-symmetric and transitive.

Lemma 2.1. *The subattribute relation is a partial order on nested attributes.* □

Informally, $M \leq N$ for $N, M \in \mathcal{NA}$ if and only if M comprises at most as much information as N does. The informal description of the subattribute relation is formally documented by the existence of a projection function $\pi^N_M : Dom(N) \to Dom(M)$ in case $M \leq N$ holds.

Definition 2.5. Let $N, M \in \mathcal{NA}$ with $M \leq N$. The *projection function* $\pi^N_M : Dom(N) \to Dom(M)$ is defined as follows:

- if $N = M$, then $\pi^N_M = id_{Dom(N)}$ is the identity on $dom(N)$,
- if $M = \lambda$, then $\pi^N_\lambda : Dom(N) \to \{ok\}$ is the constant function that maps every $v \in Dom(N)$ to ok,
- if $N = L(N_1, \ldots, N_k)$ and $M = L(M_1, \ldots, M_k)$, then $\pi^N_M = \pi^{N_1}_{M_1} \times \cdots \times \pi^{N_k}_{M_k}$ which maps every tuple $(v_1, \ldots, v_k) \in Dom(N)$ to $(\pi^{N_1}_{M_1}(v_1), \ldots, \pi^{N_k}_{M_k}(v_k)) \in Dom(M)$, and
- if $N = L[N']$ and $M = L[M']$, then $\pi^N_M : Dom(N) \to Dom(M)$ maps every list $[v_1, \ldots, v_n] \in Dom(N)$ to the list $[\pi^{N'}_{M'}(v_1), \ldots, \pi^{N'}_{M'}(v_n)] \in Dom(M)$.

□

Let $\mathcal{X}, \mathcal{Y}$ be two sets of nested attributes. $\mathcal{X}$ is called a *generalised subset* of $\mathcal{Y}$, denoted by $\mathcal{X} \subseteq_{\text{gen}} \mathcal{Y}$ if and only if for every $X \in \mathcal{X}$ there is some $Y \in \mathcal{Y}$ with $X \leq Y$. Note that $\subseteq_{\text{gen}}$ is a pre-order on sets of nested attributes.

2.3 The Brouwerian Algebra of Subattributes

Fix a set $\mathcal{U}$ of attribute names, and a set $\mathcal{L}$ of labels.

Definition 2.6. Let $N \in \mathcal{NA}$ be a nested attribute. The set $Sub(N)$ of *subattributes* of N is $Sub(N) = \{M \mid M \leq N\}$. The *bottom element* λ_N of $Sub(N)$ is given by $\lambda_N = L(\lambda_{N_1}, \ldots, \lambda_{N_k})$ whenever $N = L(N_1, \ldots, N_k)$, and $\lambda_N = \lambda$ whenever N is not a tuple-valued attribute. □

We study the algebraic structure of $Sub(N)$. A *Brouwerian Algebra* [29] is a lattice $(L, \sqsubseteq, \sqcup, \sqcap, \dot{-}, 1)$ with top element 1 and a binary operation $\dot{-}$ which satisfies $a \dot{-} b \sqsubseteq c$ iff $a \sqsubseteq b \sqcup c$ for all $c \in L$. In this case, the operation $\dot{-}$ is called the *pseudo-difference*. The *Brouwerian complement* $\neg a$ of $a \in L$ is then defined by $\neg a = 1 \dot{-} a$. A Brouwerian Algebra is also called a co-Heyting Algebra or a dual Heyting Algebra. The system of all closed subsets of a topological space is a well-known Brouwerian Algebra. It is obvious that $(Sub(N), \leq, \lambda_N, N)$ is a partially ordered set with bottom element λ_N and top element N.

Definition 2.7. Let $N \in \mathcal{NA}$ and $Y, Z \in Sub(N)$. The *join* $Y \sqcup_N Z$, *meet* $Y \sqcap_N Z$ and *pseudo difference* $Y \dot{-}_N Z$ of Y and Z in $Sub(N)$ are inductively defined as follows:

oarieli@mta.ac.il

- $Y \sqcup_N Z = Z$ iff $Y \leq Z$ iff $Y \sqcap_N Z = Y$ and $Z \dot{-}_N \lambda_N = Z$, and $Z \leq Y$ iff $Z \dot{-}_N Y = \lambda_N$,
- if $N = L[M]$, $Y = L[A], Z = L[B]$, then $Y \circ_N Z = L[A \circ_M B]$ for $\circ \in \{\sqcup, \sqcap\}$ and if $Z \not\leq Y$, then $Z \dot{-}_N Y = L[B \dot{-}_M A]$.
- if $N = L(N_1, \ldots, N_n), Y = L(A_1, \ldots, A_n)$ and $Z = L(B_1, \ldots, B_n)$, then $Y \circ_N Z = L(A_1 \circ_{N_1} B_1, \ldots, A_n \circ_{N_n} B_n)$ for $\circ \in \{\sqcup, \sqcap, \dot{-}\}$. □

In order to simplify notation, occurences of λ in a tuple-valued attribute are usually omitted if this does not cause any ambiguities. That is, the subattribute $L(M_1, \ldots, M_k) \leq L(N_1, \ldots, N_k)$ is abbreviated by $L(M_{i_1}, \ldots, M_{i_l})$ where $\{M_{i_1}, \ldots, M_{i_l}\} = \{M_j : M_j \neq \lambda_{N_j} \text{ and } 1 \leq j \leq k\}$ and $i_1 < \cdots < i_l$. If $M_j = \lambda_{N_j}$ for all $j = 1, \ldots, k$, then we use λ instead of $L(M_1, \ldots, M_k)$. The subattribute $L_1(A, \lambda, L_2[L_3(\lambda, \lambda)])$ of $L_1(A, B, L_2[L_3(C, D)])$ is abbreviated by $L_1(A, L_2[\lambda])$. However, the subattribute $L(A, \lambda)$ of $L(A, A)$ cannot be abbreviated by $L(A)$ since this may also refer to $L(\lambda, A)$.

If the context allows, we omit the index N from the operations $\sqcup_N, \sqcap_N, \dot{-}_N$ and from λ_N. The Brouwerian Algebra for $J[K(A, L[M(B, C)])]$ is illustrated in Figure 1.

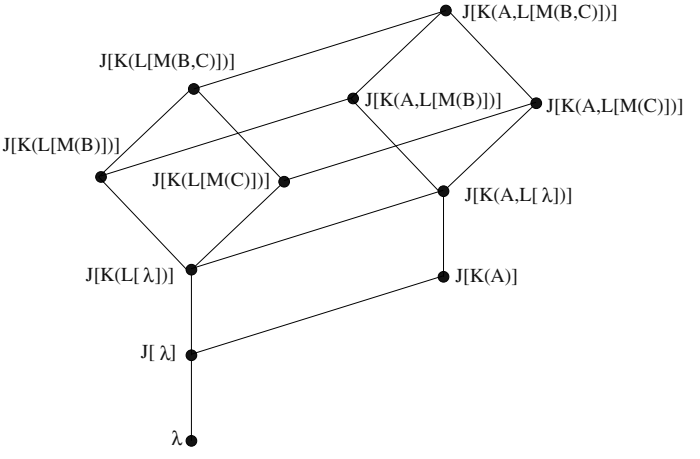


Fig. 1. The Brouwerian Algebra of $J[K(A, L[M(B, C)])]$

Given some nested attribute $N \in \mathcal{N}\mathrm{A}$ and $Y, Z \in Sub(N)$, we use $Y_N^{\mathcal{C}} = N \dot{-} Y$ to denote the *Brouwerian complement* of Y in $Sub(N)$. Again, we omit the subscript N if the context allows. The pseudo difference $Z \dot{-} Y$ of Z and Y in $Sub(N)$ satisfies

$$Z \dot{-} Y \leq X \quad \text{if and only if} \quad Z \leq Y \sqcup X$$

for all $X \in Sub(N)$. Consequently, for all $X \in Sub(N)$ holds $Y^{\mathcal{C}} \leq X$ if and only if $X \sqcup Y = N$ holds.

oarieli@mta.ac.il

The following result is straightforward to see: $Sub(\lambda)$ is isomorphic to the Boolean Algebra of order 0, $Sub(A)$, A a flat attribute, isomorphic to the Boolean Algebra of order 1. $Sub(L(P))$ is isomorphic to $Sub(P)$, $Sub(L(P_1, \ldots, P_n))$ isomorphic to the direct product of $Sub(P_1), \ldots, Sub(P_n)$, and $Sub(L[P])$ is isomorphic to $Sub(P)$ augmented by a new minimum. It is an easy exercise to show that the set of all (finite) Brouwerian Algebras is closed with respect to both operations (add a new minimum, direct product). The following theorem generalises the fact that $(\mathcal{P}(R), \subseteq, \cup, \cap, -, \emptyset, R)$ is a Boolean Algebra for a relation schema R in the RDM.

Theorem 2.1. $(Sub(N), \leq, \sqcup_N, \sqcap_N, \dot{-}_N, N)$ *forms a Brouwerian Algebra for every* $N \in \mathcal{N}\mathrm{A}$. □

Note that $(Sub(N), \leq, \sqcup, \sqcap, (\cdot)^{\mathcal{C}}, \lambda, N)$ is in general not boolean. Take for instance $N = L[A]$ and $Y = L[\lambda]$. Then $Y^{\mathcal{C}} = N$ and $Y \sqcap Y^{\mathcal{C}} = Y \neq \lambda$. Furthermore, $Y^{\mathcal{C}\mathcal{C}} = \lambda \neq Y$. Moreover, every Brouwerian Algebra is distributive.

3 Functional Dependencies

We define functional dependencies, introduce a generalisation of the Armstrong Axioms and prove that these rules are sound and complete for the implication of functional dependencies.

Definition 3.1. Let $N \in \mathcal{N}\mathrm{A}$ be a nested attribute. A *functional dependency on* N is an expression of the form $X \to Y$ where $X, Y \in Sub(N)$. A finite set $r \subseteq Dom(N)$ *satisfies* a functional dependency $X \to Y$ on N if and only if $\pi^N_Y(t_1) = \pi^N_Y(t_2)$ holds whenever $\pi^N_X(t_1) = \pi^N_X(t_2)$ for any $t_1, t_2 \in r$ holds. □

Example 3.1. Consider Pubcrawl(Person,Visit[Drink(Beer,Pub)]) and let r be

{ (Sven, [(Lübzer, Deanos), (Kindl, Highflyers)]),
(Sven, [(Kindl, Deanos), (Lübzer, Highflyers)]),
(Klaus-Dieter, [(Guiness, Irish Pub), (Speights, 3Bar),(Guiness, Irish Pub)]),
(Klaus-Dieter, [(Kölsch, Irish Pub), (Bönnsch, 3Bar), (Guiness, Irish Pub)]),
(Sebastian, [ ]) } .

It is then obvious that $\models_r$ Pubcrawl(Person) → Pubcrawl(Visit[Drink(Pub)]) holds. □

The notions of implication ($\models$) and derivability ($\vdash_{\mathfrak{R}}$) with respect to a set $\mathfrak{R}$ of inference rules for a class $\mathcal{C}$ of dependencies can be defined analogously to the notions in the RDM (see for instance [2, pp. 164-168]). In this paper, implication refers to finite implication only. Let Σ be a set of dependencies from $\mathcal{C}$ on some nested attribute N. We are interested in the set of all dependencies in $\mathcal{C}$ implied by Σ, i.e., $\Sigma^*_{\mathcal{C}} = \{\varphi \in \mathcal{C} \mid \Sigma \models \varphi\}$. Our aim is finding sets $\mathfrak{R}$ of inference rules which are sound ($\Sigma^+_{\mathcal{C}} \subseteq \Sigma^*_{\mathcal{C}}$) and complete ($\Sigma^*_{\mathcal{C}} \subseteq \Sigma^+_{\mathcal{C}}$) for the implication of dependencies in the class $\mathcal{C}$, and where $\Sigma^+_{\mathcal{C}} = \{\varphi \in \mathcal{C} \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$. We will first consider the class $\mathcal{C}$ of functional dependencies.

oarieli@mta.ac.il

Definition 3.2. The *generalised Armstrong Axioms for functional dependencies* on a nested attribute N are

$$\frac{}{X \to Y}\ Y \leq X, \quad \frac{X \to Y}{X \to X \sqcup_N Y}, \quad \frac{X \to Y,\ Y \to Z}{X \to Z}.$$

These rules are called the *reflexivity axiom*, the *extension rule* and the *transitivity rule*. □

Example 3.2. In order to sketch how these rules work we prove the soundness of the join rule

$$\frac{X \to Y,\ X \to Z}{X \to Y \sqcup Z}$$

using the following derivation tree

$$\dfrac{\dfrac{X \to Y}{X \to X \sqcup Y} \quad \dfrac{\dfrac{\dfrac{}{X \sqcup Y \to X}\ X \to Z}{X \sqcup Y \to Z}}{X \sqcup Y \to X \sqcup Y \sqcup Z}\ \dfrac{}{X \sqcup Y \sqcup Z \to Y \sqcup Z}}{X \to Y \sqcup Z}\quad .$$

□

Proposition 3.1. *The generalised Armstrong Axioms are sound.*

Proof (Sketch). It is not hard to see the soundness of the reflexivity and transitivity rule.

For the soundness of the extension rule assume $X, Y \in Sub(N)$ and let $r \subseteq Dom(N)$ with $\models_r X \to Y$. Let $t_1, t_2 \in r$ with $\pi^N_X(t_1) = \pi^N_X(t_2)$. Since $\models_r X \to Y$ we also know that $\pi^N_Y(t_1) = \pi^N_Y(t_2)$. We show by induction that also $\pi^N_{X \sqcup Y}(t_1) = \pi^N_{X \sqcup Y}(t_2)$ holds. If $X \leq Y$, then $\pi^N_{X \sqcup Y}(t_1) = \pi_Y(t_1) = \pi_Y(t_2) = \pi^N_{X \sqcup Y}(t_2)$ and similar in the case when $Y \leq X$ holds. Let $N = L(N_1, \ldots, N_n), X = L(A_1, \ldots, A_n)$ and $Y = L(B_1, \ldots, B_n)$. Since $t_1, t_2 \in Dom(N)$ it follows that $t_1 = (t_1^1, \ldots, t_1^n)$ and $t_2 = (t_2^1, \ldots, t_2^n)$ with $t_1^i, t_2^i \in Dom(N_i)$ for $i = 1, \ldots, n$. From $\pi^N_X(t_1) = \pi^N_X(t_2)$ and $\pi^N_Y(t_1) = \pi^N_Y(t_2)$ follow $\pi^{N_i}_{A_i}(t_1^i) = \pi^{N_i}_{A_i}(t_2^i)$ and $\pi^{N_i}_{B_i}(t_1^i) = \pi^{N_i}_{B_i}(t_2^i)$ for $i = 1, \ldots, n$. We conclude by hypothesis that $\pi^{N_i}_{A_i \sqcup B_i}(t_1^i) = \pi^{N_i}_{A_i \sqcup B_i}(t_2^i)$ holds for $i = 1, \ldots, n$. Then we have

$$\begin{aligned} \pi^N_{X \sqcup Y}(t_1) &= (\pi^{N_1}_{A_1 \sqcup B_1}(t_1^1), \ldots, \pi^{N_n}_{A_n \sqcup B_n}(t_1^n)) \\ &= (\pi^{N_1}_{A_1 \sqcup B_1}(t_2^1), \ldots, \pi^{N_n}_{A_n \sqcup B_n}(t_2^n)) \\ &= \pi^N_{X \sqcup Y}(t_2). \end{aligned}$$

Let $N = L[A], X = L[B]$ and $Y = L[C]$. Since $t_1, t_2 \in Dom(N)$ it follows that $t_1 = [a_1, \ldots, a_k]$ and $t_2 = [a'_1, \ldots, a'_l]$ with $a_i, a'_j \in Dom(A)$ for $i = 1, \ldots, k$ and $j = 1, \ldots, l$. From $\pi^N_X(t_1) = \pi^N_X(t_2)$ and $\pi^N_Y(t_1) = \pi^N_Y(t_2)$ follow $k = l$

oarieli@mta.ac.il

and $\pi^A_B(a_i) = \pi^A_B(a'_i)$ and $\pi^A_C(a_i) = \pi^A_C(a'_i)$ for $i = 1, \ldots, k$. We conclude by hypothesis that also $\pi^A_{B \sqcup C}(a_i) = \pi^A_{B \sqcup C}(a'_i)$ hold for $i = 1, \ldots, k$. Then we have

$$\begin{aligned} \pi^N_{X \sqcup Y}(t_1) &= [\pi^A_{B_1 \sqcup C_1}(a_1), \ldots, \pi^A_{B_k \sqcup C_k}(a_k)] \\ &= [\pi^A_{B_1 \sqcup C_1}(a'_1), \ldots, \pi^A_{B_k \sqcup C_k}(a'_k)] \\ &= \pi^N_{X \sqcup Y}(t_2). \end{aligned}$$

This concludes the proof for the soundness of the extension rule. □

Definition 3.3. Let $N \in \mathcal{NA}$. The *subattribute basis* of N is the smallest set $SubB(N) \subseteq Sub(N)$ such that for all $X \in Sub(N)$ we have $X = \sqcup Z$ for some $Z \subseteq SubB(N)$. Every $X \in SubB(N)$ is called a *basis attribute* for N. A basis attribute $X \in SubB(N)$ is called *maximal* if and only if $X \leq Y$ for some basis attribute $Y \in SubB(N)$ implies that $X = Y$ holds. Basis attributes that are not maximal are called *non-maximal*. The maximal basis attributes of N are denoted by $MaxB(N)$. □

It is immediate that $\lambda \notin SubB(N)$ since $\lambda = \sqcup\emptyset$. Furthermore, $SubB(N)$ is not an anti-chain with respect to $\leq$. It is true that $X = X^{\mathcal{CC}} \sqcup (X \sqcap X^{\mathcal{C}})$ holds in every Brouwerian Algebra. Consider now a basis attribute $Y \in SubB(N)$. If Y is maximal, then $Y \sqcap Y^{\mathcal{C}} = \lambda$ and $Y = Y^{\mathcal{CC}}$. If Y is non-maximal then $Y^{\mathcal{CC}} = \lambda$ and $Y = Y \sqcap Y^{\mathcal{C}}$. Therefore, a basis attribute Y is maximal if and only if $Y = Y^{\mathcal{CC}}$ holds, and non-maximal if and only if $Y = Y \sqcap Y^{\mathcal{C}}$ holds.

Example 3.3. Let $N = L_1(A, L_2[L_3(B, L_4[C])])$. The subattribute basis is then

$$\{L_1(A), L_1(L_2[\lambda]), L_1(L_2[L_3(L_4[\lambda])]), L_1(L_2[L_3(B)]), L_1(L_2[L_3(L_4[C])])\}.$$

The maximal basis attributes are $L_1(A), L_1(L_2[L_3(B)])$ and $L_1(L_2[L_3(L_4[C])])$. The non-maximal basis attributes are $L_1(L_2[\lambda])$ and $L_1(L_2[L_3(L_4[\lambda])])$. □

Lemma 3.1. *Let $N \in \mathcal{NA}$. There are $t_1, t_2 \in Dom(N)$ such that $\pi^N_X(t_1) \neq \pi^N_X(t_2)$ on all $\lambda \neq X \leq N$ holds.*

Proof. We prove this lemma by induction on N. For $N = \lambda$ there is nothing to show. If $N = A$ is a flat attribute, then the only $\lambda \neq X \leq N$ is $X = A$. In this case, $t_1 = a$ and $t_2 = a'$ with $a, a' \in Dom(A)$ and $a \neq a'$ are chosen. If $N = L(N_1, \ldots, N_k)$, then there are $t^i_1, t^i_2 \in Dom(N_i)$ with $\pi^{N_i}_{M_i}(t^i_1) \neq \pi^{N_i}_{M_i}(t^i_2)$ on all $\lambda \neq M_i \leq N_i$ for all $i = 1, \ldots, k$. Define $t_1 = (t^1_1, \ldots, t^k_1), t_2 = (t^1_2, \ldots, t^k_2) \in Dom(N)$. For $\lambda \neq X \leq N$ we have $X = L(M_1, \ldots, M_k)$ with $M_i \neq \lambda$ for some $i \in \{1, \ldots, k\}$. This implies that $\pi^N_X(t_1) = (\pi^{N_1}_{M_1}(t^1_1), \ldots, \pi^{N_k}_{M_k}(t^k_1)) \neq (\pi^{N_1}_{M_1}(t^1_2), \ldots, \pi^{N_k}_{M_k}(t^k_2)) = \pi^N_X(t_2)$. It remains to consider the case where $N = L[N']$. In this case we define $t_1 = [\,], t_2 = [n'] \in Dom(N)$ with $n' \in Dom(N')$. For $\lambda \neq X \leq N$ follows $X = L[M]$ with $M \leq N'$ and $\pi^N_X(t_1) = [\,] \neq [\pi^{N'}_M(n')] = \pi^N_X(t_2)$. This concludes the proof. □

Lemma 3.2. *Let $N \in \mathcal{NA}$. For all $X \in Sub(N)$ there are $t_1, t_2 \in Dom(N)$ such that $\pi^N_Y(t_1) = \pi^N_Y(t_2)$ holds if and only if $Y \leq X$.*

oarieli@mta.ac.il

Proof. If $X = \lambda$, then we apply Lemma 3.1. We assume that $X \neq \lambda$ from now on and proceed by induction on N. If $N = A$ is a flat attribute, then $X = A$ and we define $t_1 = a = t_2$ with some $a \in Dom(A)$. Consider the case where $N = L(A_1, \ldots, A_n)$ and let $X = L(B_1, \ldots, B_n)$ with $B_i \leq A_i$ for $i = 1, \ldots, n$. For $i = 1, \ldots, n$ there are $t_1^i, t_2^i \in Dom(A_i)$ with $\pi_{C_i}^{A_i}(t_1^i) = \pi_{C_i}^{A_i}(t_2^i)$ if and only if $C_i \leq B_i$ holds, by hypothesis. We define $t_1 = (t_1^1, \ldots, t_1^n)$ and $t_2 = (t_2^1, \ldots, t_2^n)$ with $t_1, t_2 \in Dom(N)$. It follows that $Y \leq X$ if and only if $Y = L(C_1, \ldots, C_n)$ with $C_i \leq B_i$ for $i = 1, \ldots, n$ if and only if $\pi_{C_i}^{A_i}(t_1^i) = \pi_{C_i}^{A_i}(t_2^i)$ for $i = 1, \ldots, n$ if and only if $\pi_Y^N(t_1) = \pi_Y^N(t_2)$. It remains to consider the case where $N = L[A]$. Consequently, $X = L[B]$ with $B \leq C$. Then there are some $t_1', t_2' \in Dom(A)$ such that $\pi_C^A(t_1') = \pi_C^A(t_2')$ if and only if $C \leq B$ by hypothesis. Defining $t_1 = [t_1'], t_2 = [t_2'] \in Dom(N)$ we infer that $\lambda \neq Y \leq X$ if and only if $Y = L[C]$ with $C \leq B$ if and only if $\pi_C^A(t_1') = \pi_C^A(t_2')$ if and only if $[\pi_C^A(t_1')] = [\pi_C^A(t_2')]$ if and only if $\pi_Y^N(t_1) = \pi_Y^N(t_2)$. The case $Y = \lambda$ is trivial. □

The following result shows that FDs can be easily captured by a natural generalisation of Armstrong's well-known axioms, if only base, record and finite list types are present. In the presence of finite set types, however, extension and join rule are only valid in a restricted form. This results in a more sophisticated set of inference rules (see [26] for details).

Theorem 3.1. *The generalised Armstrong Axioms are sound and complete for the implication of functional dependencies defined on some nested attribute N.*

Proof. It remains to show the completeness, i.e., $\Sigma^* \subseteq \Sigma^+$. Let $X \to Y \notin \Sigma^+$. We will show that $X \to Y \notin \Sigma^*$ by defining some $r \subseteq Dom(N)$ with $\models_r \Sigma^*$ and $\not\models_r X \to Y$. Let $X^+ = \sqcup\{Z \mid X \to Z \in \Sigma^+\}$. It follows that $Y \not\leq X^+$. Otherwise we had $X^+ \to Y \in \Sigma^+$ by the reflexivity axiom, $X \to X^+ \in \Sigma^+$ by the join rule and also $X \to Y \in \Sigma^+$ by the transitivity rule. Using Lemma 3.2 we define $r = \{t_1, t_2\} \subseteq Dom(N)$ by

$$\pi_Z^N(t_1) = \pi_Z^N(t_2) \qquad \text{iff} \qquad Z \leq X^+ \quad . \tag{1}$$

Since $X \leq X^+$ and $Y \not\leq X^+$ we have $\pi_X^N(t_1) = \pi_X^N(t_2)$, but $\pi_Y^N(t_1) \neq \pi_Y^N(t_2)$. This shows that $\not\models_r X \to Y$. Let $U \to V \in \Sigma$. If $U \not\leq X^+$, then $\pi_U^N(t_1) \neq \pi_U^N(t_2)$ by equation (1), and $\models_r U \to V$. If $U \leq X^+$, then $\pi_U^N(t_1) = \pi_U^N(t_2)$ by equation (1). It follows that $X^+ \to U \in \Sigma^+$ by the reflexivity axiom. From $X \to X^+, X^+ \to U, U \to V \in \Sigma^+$ follows $X \to V \in \Sigma^+$ which means that $V \leq X^+$ by definition of X^+. Again, equation (1) implies that $\pi_V^N(t_1) = \pi_V^N(t_2)$ holds. This shows $\models_r U \to V$. From $\models_r \Sigma$ follows immediately $\models_r \Sigma^*$ which completes the proof. □

4 Adding Multi-valued Dependencies

Multi-valued dependencies have been introduced in [21] and axiomatised in [9]. In this section, we will introduce multi-valued dependencies in the presence of base, record and finite list types. We will show that important properties of MVDs

oarieli@mta.ac.il

from the RDM carry over. It is in particular possible to extend the generalised Armstrong Axioms to obtain a finite axiomatisation for the class $\mathcal{C}$ of FDs and MVDs in the presence of base, record and finite list types. This axiomatisation is a natural extension of the axiomatisation in the relational case (compare for instance to [34, pp. 80,81]). A fundamental difference will be the fact that the non-trivial FD $X \to Y \sqcap Y^{\mathcal{C}}$ is implied by the MVD $X \twoheadrightarrow Y$. An axiomatisation of MVDs in the context of any advanced data models has nowhere in the literature been provided before.

4.1 Definition and First Results

Definition 4.1. Let $N \in \mathcal{N}\mathrm{A}$ be a nested attribute. A *multi-valued dependency on* N is an expression of the form $X \twoheadrightarrow Y$ where $X, Y \in Sub(N)$. A finite set $r \subseteq Dom(N)$ *satisfies* a multi-valued dependency $X \twoheadrightarrow Y$ on N if and only if for all values $t_1, t_2 \in r$ with $\pi^N_X(t_1) = \pi^N_X(t_2)$ there is a value $t \in r$ with $\pi^N_{X \sqcup Y}(t) = \pi^N_{X \sqcup Y}(t_1)$ and $\pi^N_{X \sqcup Y^C}(t) = \pi^N_{X \sqcup Y^C}(t_2)$. □

Example 4.1. Consider Pubcrawl(Person,Visit[Drink(Beer,Pub)]) from Example 3.1 again. Extend r to be

{ (Sven, [(Lübzer, Deanos), (Kindl, Highflyers)]),
(Sven, [(Kindl, Deanos), (Lübzer, Highflyers)]),
(Klaus-Dieter, [(Guiness, Irish Pub), (Speights, 3Bar),(Guiness, Irish Pub)]),
(Klaus-Dieter, [(Kölsch, Irish Pub), (Bönnsch, 3Bar), (Guiness, Irish Pub)]),
(Klaus-Dieter, [(Guiness, Highflyers), (Speights, Deanos), (Guiness, 3Bar)]),
(Klaus-Dieter, [(Kölsch, Highflyers), (Bönnsch, Deanos), (Guiness, 3Bar)]),
(Sebastian, [ ]) }

Obviously, Pubcrawl(Person) → Pubcrawl(Visit[Drink(Pub)]) is not satisfied by r, and neither is Pubcrawl(Person) → Pubcrawl(Visit[Drink(Beer)]). However, $\models_r$ Pubcrawl(Person) ↠ Pubcrawl(Visit[Drink(Pub)]). This MVD informally says that a person has prefered lists of pubs, e.g. according to the weekday, and prefered lists of beers, e.g. according to the mood that person is in. Since a weekday is independent from the mood of a person, all possible combinations of these lists can occur. Note that $\models_r$ Pubcrawl(Person) → Pubcrawl(Visit[λ]) holds. This means informally that the person determines the number of bars visited by that person. □

A dependency σ on some nested attribute N is called trivial if and only if $\models_r \sigma$ for every $r \in Dom(N)$. The following result characterises trivial MVDs.

Lemma 4.1. *Let $N \in \mathcal{N}A$ and $X \twoheadrightarrow Y$ a multi-valued dependency on N. Then is $X \twoheadrightarrow Y$ trivial if and only if $Y \leq X$ or $X \sqcup Y = N$.*

Proof. We show first that $X \twoheadrightarrow Y$ is trivial, if $Y \leq X$ or $X \sqcup Y = N$. Let $r \subseteq Dom(N)$ and $t_1, t_2 \in r$ with $\pi^N_X(t_1) = \pi^N_X(t_2)$. If there is some $t \in r$ with $\pi^N_{X \sqcup Y}(t) = \pi^N_{X \sqcup Y}(t_1)$ and $\pi^N_{X \sqcup Y^C}(t) = \pi^N_{X \sqcup Y^C}(t_2)$, then $\models_r X \twoheadrightarrow Y$. If $Y \leq X$, then take $t = t_2$. If $X \sqcup Y = N$, or equivalently $Y^{\mathcal{C}} \leq X$, then take $t = t_1$.

oarieli@mta.ac.il

Let now be $Y \not\leq X$ and $X \sqcup Y \neq N$, i.e., $Y^{\mathcal{C}} \not\leq X$. We show that there is some $r \subseteq Dom(N)$ with $\not\models_r X \twoheadrightarrow Y$. Define $r = \{t_1, t_2\}$ by

$$\pi^N_Z(t_1) \quad = \quad \pi^N_Z(t_2) \qquad \text{iff} \qquad Z \quad \leq \quad X$$

using Lemma 3.2. Now $\models_r X \twoheadrightarrow Y$, if there is some $t \in r$ with $\pi^N_{X \sqcup Y}(t) = \pi^N_{X \sqcup Y}(t_1)$ and $\pi^N_{X \sqcup Y^C}(t) = \pi^N_{X \sqcup Y^C}(t_2)$. If $t = t_1$, then the second condition is violated since $Y^C \not\leq X$. If $t = t_2$, then the first condition is violated since $Y \not\leq X$. Hence, $\not\models_r X \twoheadrightarrow Y$. □

Fagin proves in [21] that MVDs "provide a necessary and sufficient condition for a relation to be decomposable into two of its projections without loss of information (in the sense that the original relation is guaranteed to be the join of the two projections)."

Definition 4.2. Let $N \in \mathcal{NA}$ and $X, Y \in Sub(N)$. Let $r_1 \subseteq Dom(X)$ and $r_2 \subseteq Dom(Y)$. Then

$$r_1 \bowtie r_2 \quad = \quad \{t \in Dom(X \sqcup Y) \mid \text{ there are } t_1 \in r_1, t_2 \in r_2 \text{ with } \pi^{X \sqcup Y}_X(t) = t_1 \text{ and } \pi^{X \sqcup Y}_Y(t) = t_2\}.$$

is called the *generalised join* $r_1 \bowtie r_2$ *of* r_1 and r_2. □

We will now prove that MVDs still have the same property in the presence of base, record and finite list types. The *projection* $\pi_X(r)$ of $r \subseteq Dom(N)$ on $X \leq N$ is defined as $\{\pi^N_X(t) \mid t \in r\}$. In this sense, $r \subseteq Dom(N)$ satisfies the MVD $X \twoheadrightarrow Y$ exactly when r is the lossless generalised join of its projections on $X \sqcup Y$ and $X \sqcup Y^{\mathcal{C}}$, i.e., $r = \pi_{X \sqcup Y}(r) \bowtie \pi_{X \sqcup Y^C}(r)$.

Theorem 4.1. *Let $N \in \mathcal{NA}$, $r \subseteq Dom(N)$ and $X \twoheadrightarrow Y$ a multi-valued dependency on N. Then is $X \twoheadrightarrow Y$ satisfied by r if and only if $r = \pi_{X \sqcup Y}(r) \bowtie \pi_{X \sqcup Y^C}(r)$.*

Proof. Let $r_1 = \pi_{X \sqcup Y}(r)$ and $r_2 = \pi_{X \sqcup Y^C}(r)$. Note that $r \subseteq r_1 \bowtie r_2$ is always satisfied.

First, let $\models_r X \twoheadrightarrow Y$. We show that $r_1 \bowtie r_2 \subseteq r$. Let $t \in r_1 \bowtie r_2$. Then there are $t_1 \in r_1$ and $t_2 \in r_2$ with

$$\pi^N_X(t) = \pi^{X \sqcup Y}_X(t_1) = \pi^{X \sqcup Y^{\mathcal{C}}}_X(t_2), \quad \pi^N_Y(t) = \pi^{X \sqcup Y}_Y(t_1), \quad \pi^N_{Y^C}(t) = \pi^{X \sqcup Y}_{Y^C}(t_2).$$

Since $t_1 \in r_1$, there is some $t'_1 \in r$ with $\pi^N_{X \sqcup Y}(t'_1) = t_1$. Correspondingly, since $t_2 \in r_2$, there is some $t'_2 \in r$ with $\pi^N_{X \sqcup Y^C}(t'_2) = t_2$. From $\pi^N_X(t'_1) = \pi^N_X(t'_2)$ and $\models_r X \twoheadrightarrow Y$ follows the existence of some $t_3 \in r$ with $\pi^N_{X \sqcup Y}(t_3) = \pi^N_{X \sqcup Y}(t'_1) = t_1$ and $\pi^N_{X \sqcup Y^C}(t_3) = \pi^N_{X \sqcup Y^C}(t'_2) = t_2$. It follows that $t = t_3 \in r$ and, therefore, $r_1 \bowtie r_2 \subseteq r$.

Let now $r = r_1 \bowtie r_2$ and $t_1, t_2 \in r$ with $\pi^N_X(t_1) = \pi^N_X(t_2)$. Let $t'_1 \in r_1$ with $t'_1 = \pi^N_{X \sqcup Y}(t_1)$ and $t'_2 \in r_2$ with $t'_2 = \pi^N_{X \sqcup Y^C}(t_2)$. Since $r_1 \bowtie r_2 \subseteq r$, there is some $t \in r$ with $\pi^N_{X \sqcup Y}(t) = t'_1$ and $\pi^N_{X \sqcup Y^C}(t) = t'_2$. This shows $\models_r X \twoheadrightarrow Y$. □

oarieli@mta.ac.il

4.2 Sound Inference Rules

A sound and complete set of inference rules for FDs and MVDs has been provided in [9]. We will show in this section that natural extensions of the (sound and complete) rules from [34, p.80,81] are also sound in the presence of base, record and finite list types. Apart from these rules there is a further sound rule which allows to derive a non-trivial FD $X \to Y \sqcap Y^{\mathcal{C}}$ from an MVD $X \twoheadrightarrow Y$.

Proposition 4.1. *The following inference rules*

$$\frac{}{X \to Y}\ Y \leq X \quad \textit{(reflexivity axiom)} \qquad \frac{X \to Y}{X \to X \sqcup Y} \quad \textit{(extension rule)}$$

$$\frac{X \to Y,\ Y \to Z}{X \to Z} \quad \textit{(transitivity rule)} \qquad \frac{X \to Y}{X \twoheadrightarrow Y} \quad \textit{(implication rule)}$$

$$\frac{X \twoheadrightarrow Y}{X \twoheadrightarrow Y^{\mathcal{C}}} \quad \textit{(Brouwerian-complement rule)} \qquad \frac{X \twoheadrightarrow Y}{W \sqcup X \twoheadrightarrow V \sqcup Y}\ V \leq W \quad \textit{(multi-valued augmentation rule)}$$

$$\frac{X \twoheadrightarrow Y \quad Y \to Z}{X \twoheadrightarrow (Z \dot{-} Y)} \quad \textit{(pseudo-transitivity rule)} \qquad \frac{X \twoheadrightarrow Y \quad Y \to Z}{X \to (Z \dot{-} Y)} \quad \textit{(mixed pseudo-transitivity rule)}$$

$$\frac{X \twoheadrightarrow Y \quad X \twoheadrightarrow Z}{X \twoheadrightarrow (Y \sqcup Z)} \quad \textit{(multi-valued join rule)} \qquad \frac{X \twoheadrightarrow Y \quad X \twoheadrightarrow Z}{X \twoheadrightarrow (Z \dot{-} Y)} \quad \textit{(pseudo-difference rule)}$$

$$\frac{X \twoheadrightarrow Y}{X \to Y \sqcap Y^{\mathcal{C}}} \quad \textit{(mixed meet rule)} \qquad \frac{X \twoheadrightarrow Y \quad X \twoheadrightarrow Z}{X \twoheadrightarrow (Y \sqcap Z)} \quad \textit{(multi-valued meet rule)}$$

are sound for the implication of functional and multi-valued dependencies.

Proof (Sketch). The soundness proofs are lengthy, but can be carried over from the RDM using the algebraic framework of a Brouwerian Algebra. We will prove the soundness of the mixed meet rule.

Let $t_1, t_2 \in r$ with $\pi^N_X(t_1) = \pi^N_X(t_2)$. Applying the premise gives us some $t \in r$ with $\pi^N_{X \sqcup Y}(t) = \pi^N_{X \sqcup Y}(t_1)$ and $\pi^N_{X \sqcup Y^{\mathcal{C}}}(t) = \pi^N_{X \sqcup Y^{\mathcal{C}}}(t_2)$. As $Y \sqcap Y^{\mathcal{C}} \leq Y, Y^{\mathcal{C}}$ holds by definition of the meet we derive

$$\pi^N_{Y \sqcap Y^{\mathcal{C}}}(t_1) = \pi^N_{Y \sqcap Y^{\mathcal{C}}}(t) = \pi^N_{Y \sqcap Y^{\mathcal{C}}}(t_2)$$

which proves $\models_r X \to Y \sqcap Y^{\mathcal{C}}$. □

It is easy to see that all rules from Propostion 4.1 except the mixed meet rule are natural extensions of rules in the RDM (compare [34, p. 80,81]). Interpreting the mixed meet rule in relational databases means that the trivial FD $X \to \emptyset$ can be derived from the MVD $X \twoheadrightarrow Y$, and is therefore not needed.

In what follows, it is important to emphasize the importance of the mixed meet rule. It says informally that $\models_r X \twoheadrightarrow Y$ implies that two elements of r

oarieli@mta.ac.il

which are coincident on X will also coincide on all basis attributes in $SubB(Y)$ which are not maximal basis attributes of N on which the MVD is defined.

Example 4.2. Consider again Example 4.1. The r given there satisfies the MVD

$$\text{Pubcrawl(Person)} \twoheadrightarrow \text{Pubcrawl(Visit[Drink(Pub)])}.$$

According to the mixed meet rule this implies also that r satisfies the FD

$$\begin{aligned}\text{Pubcrawl(Person)} \rightarrow \text{Pubcrawl(Visit[Drink(Pub)])} \sqcap \\ \text{Pubcrawl(Visit[Drink(Beer)])}\end{aligned}$$

which is $\text{Pubcrawl(Person)} \rightarrow \text{Pubcrawl(Visit}[\lambda])$. □

4.3 Dependency Basis

Consider the set of all Y with $X \twoheadrightarrow Y \in \Sigma^+$ for a fixed X defined on some nested attribute N. According to the multi-valued join, multi-valued meet and pseudo-difference rule this set, partially ordered by $\leq$, forms a Brouwerian Algebra. Due to the mixed meet rule, all basis attributes of Y which are not maximal in N are already functionally determined by X. Attributes $Y^{\mathcal{CC}} \neq \lambda$ with $X \twoheadrightarrow Y \in \Sigma^+$ which are $\leq$-minimal with this property are therefore of great interest.

Definition 4.3. Let $N \in \mathcal{NA}$, $X \in Sub(N)$ and Σ a set of multi-valued and functional dependencies on N. Let $Dep(X)$ be the set of all $Y \in Sub(N)$ with $X \twoheadrightarrow Y \in \Sigma^+$ and $X^+ = \sqcup\{Y \mid X \rightarrow Y \in \Sigma^+\}$. Let $X^M \subseteq Sub(N)$ have the following properties:

1. for all $U \in MaxB(N)$ there is a unique $V \in X^M$ with $U \leq V$,
2. for all $U \in X^M$ there is some $W \subseteq MaxB(N)$ with $U = \sqcup W$,
3. for all $V \in Dep(X)$ there is some $Z \subseteq X^M$ with $V^{\mathcal{CC}} = \sqcup Z$, and
4. X^M is maximal with these properties with respect to $\subseteq_{\text{gen}}$.

The *dependency basis* of X with respect to Σ is $DepB(X) = SubB(X^+) \cup X^M$. □

Note that $\{MaxB(W) \mid W \in X^M\}$ is the partition of $MaxB(N)$ which is generated by $\{MaxB(Y^{\mathcal{CC}}) \mid Y \in Dep(X)\}$. The first property says that every maximal basis attribute of N is the subattribute of exactly one element in X^M. The second property guarantees that every element in X^M is the join of maximal basis attributes of N. If $X \twoheadrightarrow V \in \Sigma^+$ holds, then the join of all basis attributes in V which are maximal in N (, i.e. $V^{\mathcal{CC}}$,) is the join over elements of X^M by the third property. The last property guarantees the uniqueness of the dependency basis and that $X \twoheadrightarrow W \in \Sigma^+$ holds for all $W \in X^M$.

Lemma 4.2. *Let $N \in \mathcal{NA}$, Σ a set of functional and multi-valued dependencies on N, $X \leq N$ and $DepB(X) = SubB(X^+) \cup X^M$ the dependency basis of X with respect to Σ. If $W \in X^M$, then $W \in Dep(X)$.*

oarieli@mta.ac.il

Proof (Sketch). Let $W \in X^M$ and assume $W \notin Dep(X)$. Let $\overline{W}$ be the $\leq$-smallest superattribute of W with $\overline{W} \in Dep(X)$. We can assume that $\overline{W} = \overline{W}^{\mathcal{CC}}$ holds due to the Brouwerian-complement rule. According to Definition 4.3 we have $\overline{W} = W \sqcup W_1 \sqcup \cdots \sqcup W_n$ with $W, W_1, \ldots, W_n \in X^M$ and $n \geq 1$. It can then be shown that $X^N = X^M \backslash \{W, W_1, \ldots, W_n\} \cup \{\overline{W}\}$ satisfies the first three properties of Definition 4.3. This contradicts the maximality of X^M with respect to $\subseteq_{gen}$ and $W \in Dep(X)$ follows. □

We can now show that an MVD $X \twoheadrightarrow Y$ is derivable from Σ if and only if the right-hand side Y is the join over some elements of the dependency basis of X with respect to Σ.

Proposition 4.2. *Let $N \in \mathcal{N}\mathrm{A}$ and Σ as set of functional and multi-valued dependencies on N. Then*

1. *$X \twoheadrightarrow Y \in \Sigma^+$ if and only if $Y = \sqcup Z$ for some $Z \subseteq DepB(X)$*
2. *$X \to Y \in \Sigma^+$ if and only if $Y \leq X^+$.*

Proof. The second property is obvious. Let $Y \in Dep(X)$. Recall that $Y = Y^{\mathcal{CC}} \sqcup (Y \sqcap Y^{\mathcal{C}})$ holds. From $Y \in Dep(X)$ follows $Y^{\mathcal{CC}} = \sqcup Z_1$ for some $Z_1 \subseteq X^M$ by Definition 4.3, and $X \to Y \sqcap Y^{\mathcal{C}} \in \Sigma^+$ by the mixed meet rule. It follows that $Y \sqcap Y^{\mathcal{C}} \leq X^+$ and, therefore, $Y \sqcap Y^{\mathcal{C}} = \sqcup Z_2$ for some $Z_2 \subseteq SubB(X^+)$. Hence, $Y = \sqcup Z$ for some $Z \subseteq DepB(X)$.

Assume now that $Y = \sqcup Z$ holds for some $Z \subseteq DepB(X)$. Then $Z = Z_1 \cup Z_2$ with $Z_1 \subseteq SubB(X^+)$ and $Z_2 \subseteq X^M$. It follows that $\sqcup Z_1 = Y_1 \leq X^+$, and the reflexivity rule, join rule and transitivity rule imply $X \to Y_1 \in \Sigma^+$. The implication rule gives $X \twoheadrightarrow Y_1 \in \Sigma^+$. Furthermore, if $Z_2 = \{V_1, \ldots, V_m\} \subseteq X^M$, then $X \twoheadrightarrow V_i \in \Sigma^+$ for $1 \leq i \leq m$ by Lemma 4.2. Applying the multi-valued join rule gives $X \twoheadrightarrow Y \in \Sigma^+$. □

4.4 Completeness

Proving the completeness result for functional and multi-valued dependencies will involve the definition of some instance which satisfies all dependencies in Σ. This instance will initially contain two elements t_1, t_2 which are coincident on exactly all attributes which are functionally determind by some fixed X. Afterwards new elements are generated and added to the instance by exhaustively combining values from t_1 on some $W \subseteq X^M$ and the values from t_2 on $X^M \backslash W$. Let $W, W' \in X^M$. Since the meet $W \sqcap W'$ is not necessarily equal to λ one needs to show that such a construction is possible in general. It will turn out that $SubB(W \sqcap W')$ contains only attributes already functionally determined by X.

Definition 4.4. Let $N \in \mathcal{N}\mathrm{A}$, $X' \subseteq MaxB(N)$ and $X = \sqcup X'$. A basis attribute $Y \in SubB(X)$ is *possessed by* X if and only if every basis attribute $Z \in SubB(N)$ with $Y \leq Z$ is also a subattribute of X ($Z \leq X$). □

It follows that $SubB(W \sqcap W')$ with $W, W' \in X^M$ contains only basis attributes of W or W' which are neither possessed by W nor by W'.

oarieli@mta.ac.il

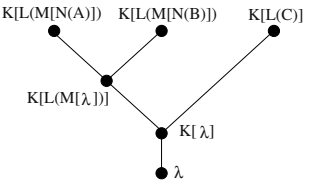


Fig. 2. The subattribute basis of $K[L(M[N(A,B)],C)]$

Example 4.3. Let $K[L(M[N(A,B)],C)] \in \mathcal{N}\mathrm{A}$, and $X = K[L(M[N(A,B)])]$. Then X does possess $K[L(M[\lambda])]$, but does *not* possess $K[\lambda]$. For an illustration see also Figure 2. □

A basis attribute is not possessed by some X exactly if it is also a basis attribute of $X^{\mathcal{C}}$. According to the mixed meet rule it follows that basis attributes which are not possessed by some element in X^M are functionally determined by X.

Lemma 4.3. *Let $N \in \mathcal{N}A$, $X' \subseteq MaxB(N)$, $X = \sqcup X'$ and $Y \in SubB(X)$. Then is Y possessed by X if and only if $Y \notin SubB(X^{\mathcal{C}})$.* □

Corollary 4.1. *Let $N \in \mathcal{N}A$, $X \leq N$, Σ a set of functional and multi-valued dependencies on N, and $DepB(X) = SubB(X^+) \cup X^M$. Then for every $W \in X^M$ and every $Y \in SubB(W)$ that is not possessed by W follows that $Y \leq X^+$.*

Proof. Since Y is not possessed by W, $Y \in SubB(W^{\mathcal{C}})$ by Lemma 4.3 and therefore $Y \leq W \sqcap W^{\mathcal{C}}$. Lemma 4.2 implies that $X \twoheadrightarrow W \in \Sigma^+$ holds, and using the mixed meet rule we infer $X \to W \sqcap W^{\mathcal{C}}$. The reflexivity rule implies $W \sqcap W^{\mathcal{C}} \to Y \in \Sigma^+$ from $Y \leq W \sqcap W^{\mathcal{C}}$. The statement $X \to Y \in \Sigma^+$ follows now from the transitivity rule. □

Suppose $DepB(X) = SubB(X^+) \cup \{W_{0,1}, \ldots, W_{0,m}, W_1, \ldots, W_k\}$ with $W_{0,i} \leq X^+$ for $i = 1, \ldots, m$ and $W_1, \ldots, W_k \not\leq X^+$. We have seen that $SubB(W_i \sqcap W_j)$, $i \neq j$, contains only basis attributes of W_i or W_j neither possessed by W_i nor by W_j. It follows that $X \to W_i \sqcap W_j$ holds.

Assume now that there are two elements $t_1, t_2 \in Dom(N)$ which coincide on at least all subattributes of X^+. It is then easy to see that one can substitute the values of t_1 on all subattributes of some given $W_i \in X^M$ for the corresponding values of t_2 and end up with an element in $Dom(N)$.

Lemma 4.4. *Let $N \in \mathcal{N}A$, Σ a set of functional and multi-valued dependencies on N and $X \leq N$. Let $DepB(X) = SubB(X^+) \cup X^M$ with $X^M = \{W_{0,1}, \ldots, W_{0,m}, W_1, \ldots, W_k\}$ and $W_{0,i} \leq X^+$ for $1 \leq i \leq m$ and $W_1, \ldots, W_k \not\leq X^+$. Let $t_1, t_2 \in Dom(N)$ with $\pi^N_W(t_1) = \pi^N_W(t_2)$, if $W \leq X^+$. Then for all $W = \sqcup W'$ with $W' \subseteq \{W_1, \ldots, W_k\}$ there is some $t \in Dom(N)$ with $\pi^N_W(t) = \pi^N_W(t_1)$ and $\pi^N_{W^C}(t) = \pi^N_{W^C}(t_2)$.* □

Corollary 4.1 shows that every basis attribute which is not possessed by any $W_i \in X^M$ is functionally determined by X, i.e., elements in $Dom(N)$ with the

oarieli@mta.ac.il

same value on X will also coincide on all basis attributes which are not possessed by any $W_i \in X^M$. The statement from Lemma 4.4 is therefore equivalent to the fact that there is some $t \in Dom(N)$ with

$$\pi_A^N(t) = \begin{cases} \pi_A^N(t_1) & \text{, if } A \text{ is possessed by some } W_i \in W' \\ \pi_A^N(t_2) & \text{, else} \end{cases} .$$

Theorem 4.2. *The set of rules from Proposition 4.1 is sound and complete for the implication of functional and multi-valued dependencies on some nested attribute N.*

Proof. Due to Proposition 4.1, it remains to show the completeness, i.e., $\Sigma^* \subseteq \Sigma^+$. Let $X \in Sub(N)$. Let $DepB(X) = SubB(X^+) \cup X^M$ with $X^M = \{W_{0,1}, \ldots, W_{0,m}, W_1, \ldots, W_k\}$ and $W_{0,i} \leq X^+$ for $i = 1, \ldots, m$ and $W_1, \ldots, W_k \not\leq X^+$. Take $t_1, t_2 \in Dom(N)$ defined by

$$\pi_W^N(t_1) = \pi_W^N(t_2) \qquad \text{iff} \qquad W \leq X^+.$$

Recall that such t_1, t_2 exist according to Lemma 3.2. Define an instance $r \subseteq Dom(N)$ with $t_1, t_2 \in r$ and add for every $W = \sqcup W'$ with $W' \subseteq \{W_1, \ldots, W_k\}$ the $t \in Dom(N)$ with $\pi_W^N(t) = \pi_W^N(t_1)$ and $\pi_{W^c}^N(t) = \pi_{W^c}^N(t_2)$ from Lemma 4.4 to r. Obviously, r has exactly 2^k elements, and if $\pi_{W_l}^N(t_i) \neq \pi_{W_l}^N(t_j)$, then also $\pi_W^N(t_i) \neq \pi_W^N(t_j)$ on all $W \leq W_l$ which are possessed by W_l.
We will show that $\models_r \Sigma$. Then, for $X \to Y \in \Sigma^*$ we have $\models_r X \to Y$. Since all elements of r coincide on X^+, r can only satisfy $X \to Y$ if all elements of r also coincide on Y. It follows by construction of r that $Y \leq X^+$. Proposition 4.2 implies $X \to Y \in \Sigma^+$. For $X \twoheadrightarrow Y \in \Sigma^*$ we have $\models_r X \twoheadrightarrow Y$. Again by construction, r can only satisfy $X \twoheadrightarrow Y$ if $Y = X_0 \sqcup W_{i_1} \sqcup \cdots \sqcup W_{i_m}$ with $X_0 \leq X^+$ and $1 \leq i_1 < \cdots < i_m \leq k$. Therefore, $X \twoheadrightarrow Y \in \Sigma^+$ by Proposition 4.2. We now show that $\models_r \Sigma$ holds.

1. Suppose $U \to V \in \Sigma$. Define

$$W = \sqcup\{W_i \mid \exists U'.U' \leq U \text{ and } U' \text{ is possessed by } W_i\}.$$

It follows that $U \leq X^+ \sqcup W$ since every subattribute of U that is possessed by some W_i is also a subattribute of W and every subattribute of U that is not possessed by any W_i is a subattribute of X^+. The reflexivity rule implies $X^+ \sqcup W \to U \in \Sigma^+$. Using the transitivity rule gives $X^+ \sqcup W \to V \in \Sigma^+$. Take $t_i, t_j \in r$ with $\pi_U^N(t_i) = \pi_U^N(t_j)$. All elements of r are equal on X^+. Assume $\pi_W^N(t_i) \neq \pi_W^N(t_j)$. Then there is some W_l with $\pi_{W_l}^N(t_i) \neq \pi_{W_l}^N(t_j)$ and some subattribute $U' \leq U$ which is possessed by W_l. Consequently, $\pi_{U'}^N(t_i) \neq \pi_{U'}^N(t_j)$ and, therefore, $\pi_U^N(t_i) \neq \pi_U^N(t_j)$ too, a contradiction. It follows that $\pi_W^N(t_i) = \pi_W^N(t_j)$ holds and therefore $\pi_{X^+ \sqcup W}^N(t_i) = \pi_{X^+ \sqcup W}^N(t_j)$ as well. Now, $X^+ \sqcup W$ is the join of elements in $DepB(X)$, i.e., $X \twoheadrightarrow X^+ \sqcup W \in \Sigma^+$ by Proposition 4.2. Hence, we infer $X \to (V \dot{-} (X^+ \sqcup W)) \in \Sigma^+$ by the mixed pseudo-transitivity rule. Proposition 4.2 implies $V \dot{-} (X^+ \sqcup W) \leq X^+$, and therefore $\pi_{V \dot{-} (X^+ \sqcup W)}^N(t_i) = \pi_{V \dot{-} (X^+ \sqcup W)}^N(t_j)$. Since $V \leq (X^+ \sqcup W) \sqcup (V \dot{-} (X^+ \sqcup W))$ holds, we obtain $\pi_V^N(t_i) = \pi_V^N(t_j)$. This proves $\models_r U \to V$.

oarieli@mta.ac.il

2. Suppose $U \twoheadrightarrow V \in \Sigma$. Define again

$$W = \sqcup\{W_i \mid \exists U'.U' \leq U \text{ and } U' \text{ is possessed by } W_i\}.$$

As before, $U \leq X^+ \sqcup W$ holds. From $U \twoheadrightarrow V \in \Sigma$ and $\lambda \leq X^+ \sqcup W$ follows $X^+ \sqcup W \twoheadrightarrow V \in \Sigma^+$ by the multi-valued augmentation rule.
Take $t_i, t_j \in r$ with $\pi^N_U(t_i) = \pi^N_U(t_j)$. Again, the construction of r implies $\pi^N_{X^+ \sqcup W}(t_i) = \pi^N_{X^+ \sqcup W}(t_j)$. Since $X \twoheadrightarrow X^+ \sqcup W \in \Sigma^+$ holds by Proposition 4.2, the pseudo-transitivity rule allows to derive $X \twoheadrightarrow (V \dot{-} (X^+ \sqcup W)) \in \Sigma^+$. Therefore, $V \dot{-} (X^+ \sqcup W)$ is the join of some elements in $DepB(X)$ by Proposition 4.2. By construction of r there is some $t \in r$ with

$$\pi^N_{X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))}(t) = \pi^N_{X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))}(t_i)$$

and

$$\pi^N_{X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))^{\mathcal{C}}}(t) = \pi^N_{X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))^{\mathcal{C}}}(t_j).$$

As $U, V \leq X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))$ hold, we have $U \sqcup V \leq X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))$ and therefore $\pi^N_{U \sqcup V}(t) = \pi^N_{U \sqcup V}(t_i)$. Since

$$(V \dot{-} (X^+ \sqcup W))^{\mathcal{C}\mathcal{C}} \leq V \dot{-} (X^+ \sqcup W) \leq V,$$

it follows from the defining property of the Brouwerian-complement that $V \sqcup (V \dot{-} (X^+ \sqcup W))^{\mathcal{C}} = N$, and consequently $V^{\mathcal{C}} \leq (V \dot{-} (X^+ \sqcup W))^{\mathcal{C}}$. But then $U \sqcup V^{\mathcal{C}} \leq X^+ \sqcup W \sqcup (V \dot{-} (X^+ \sqcup W))^{\mathcal{C}}$ and thus $\pi^N_{U \sqcup V^{\mathcal{C}}}(t) = \pi^N_{U \sqcup V^{\mathcal{C}}}(t_j)$. This proves $\models_r U \twoheadrightarrow V$. □

In summary, the construction is based on the relational theory for maximal basis attributes and the fact that non-maximal basis attributes not possessed by any $W \in X^M$ are functionally determined by X. The rest follows by using the algebraic framework.

5 Future Work

An examination of the independence of the inference rules from Proposition 4.1 was beyond the scope of this paper. In particular, the role of the Brouwerian-complement rule will be similar to the one of the complementation rule in relational databases [11,30]. Removing the Brouwerian-complement rule the completeness proof of Theorem 4.2 does not apply. We are confident that the results from [12] can be carried over. Several lines of research might be followed. Next we would like to investigate the finite implication problem of FDs and MVDs extending the work in [6] to advanced data models. Normalisation and its semantic justification should be studied in the presence of FDs and MVDs leading to an extension of fourth normal form [21,42]. Other classes of relational dependencies (see [39] for an overview), such as join and inclusion dependencies, are to be generalised in the future. Finally, various combinations of different types including sets, multisets, unions and references are objects to future research.

oarieli@mta.ac.il

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. M. Arenas and L. Libkin. A normal form for XML documents. In *PODS 2002*. ACM, 2002.
4. W. W. Armstrong. Dependency structures of database relationships. *Information Processing*, pages 580–583, 1974.
5. C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin Cummings, 1992.
6. C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *Transactions on Database Systems*, 5(3):241–259, 1980.
7. C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemata. In *Transactions on Database Systems*, pages 30–59. Association for Computing Machinery, 1979.
8. C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *Proceedings of 4th International Conference on Very Large Databases*, pages 113–124, 1978.
9. C. Beeri, R. Fagin, and J. H. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In *Proceedings of the International Conference on Management of Data*, pages 47–61. Association for Computing Machinery, 1977.
10. P. A. Bernstein and N. Goodman. What does Boyce-Codd normal form do? In *Proceedings of the International Conference on Very Large Databases*, pages 245–259, 1980.
11. J. Biskup. On the complementation rule for multivalued dependencies in database relations. *Acta Informatica*, 10(3):297–305, 1978.
12. J. Biskup. Inferences of multivalued dependencies in fixed and undetermined universes. *Theoretical Computer Science*, 10(1):93–106, 1980.
13. J. Biskup. Database schema design theory:achievements and challenges. In *Proceedings of the 6th International Conference on Information Systems and Management of Data*, number 1066 in Lecture Notes in Computer Science, pages 14–44. Springer, 1995.
14. J. Biskup. Achievements of relational database schema design theory revisited. In *Semantics in databases*, number 1358 in Lecture Notes in Comptuer Science, pages 29–54. Springer, 1998.
15. F. Bry and P. Kröger. A computational biology database digest: data, data analysis, and data management. *Distributed and Parallel Databases*, 13(1):7–42, 2003.
16. P. P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions Database Systems 1*, pages 9–36, 1976.
17. P. P. Chen. English sentence structure and entity-relationship diagrams. *Information Science 29*, pages 127–149, 1983.
18. E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, pages 377–387, 1970.
19. E. F. Codd. Further normalization of the database relational model. In *Courant Computer Science Symposia 6: Data Base Systems*, pages 33–64. Prentice-Hall, 1972.

oarieli@mta.ac.il

20. E. F. Codd. Recent investigations in relational database system. In *Proceedings of the IFIP Conference*, pages 1017–1021, 1974.
21. R. Fagin. Multivalued dependencies and a new normal form for relational databases. *Association for Computing Machinery*, 2(3):262–278, 1977.
22. R. Fagin. A normal form for relational databases that is based on domains and keys. *Transactions on Database Systems*, pages 387–415, 1981.
23. G. Gardarin, J.-P. Cheiney, G. Kiernan, D. Pastre, and H. Stora. Managing complex objects in an extensible relational dbms. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases, August 22–25, 1989, Amsterdam, The Netherlands*, pages 55–65. Morgan Kaufmann, 1989.
24. S. Hartmann. Decomposing relationship types by pivoting and schema equivalence. *Data & Knowledge Engineering*, 39:75–99, 2001.
25. S. Hartmann and S. Link. More functional dependencies for xml. *Lecture Notes in Computer Science*, 2798:355–369, 2003.
26. S. Hartmann and S. Link. On functional dependencies in advanced data models. *Electronic Notes in Theoretical Computer Science*, 84, 2003.
27. S. Hartmann, S. Link, and K.-D. Schewe. Generalizing Boyce-Codd normal form to conceptual databases. In *Pre-Proceedings of the 13th European-Japanese Conference on Information Modeling and Knowledge Bases*, pages 93–110, 2003.
28. R. Hull and R. King. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3), 1987.
29. J. McKinsey and A. Tarski. On closed elements in closure algebras. *Annals of Mathematics*, 47:122–146, 1946.
30. A. Mendelzon. On axiomatising multivalued dependencies in relational databases. *Journal of the ACM*, 26(1):37–44, 1979.
31. W. Y. Mok, Y. K. Ng, and D. W. Embley. A normal form for precisely charachterizing redundancy in nested relations. *Transactions on Database Systems*, 21:77–106, 1996.
32. S. Naqvi and S. Tsur. *A logical language for data and knowledge bases*. Computer Science Press, 1989.
33. Z. M. Özsoyoglu and L. Y. Yuan. A new normal form for nested relations. *Transactions on Database Systems*, 12:111–136, 1987.
34. J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model*. Springer-Verlag, 1989.
35. J. Richardson. Supporting lists in a datamodel. In *Proceeding of VLDB*, pages 127–192, 1992.
36. K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(4):49–85, 1993.
37. P. Seshadri, M. Livny, and R. Ramakrishnan. The design and implementation of sequence database system. In *Proceedings of the Twentysecond International Conference on Very Large Data Bases, Mumbai, India*, 1996.
38. Z. Tari, J. Stokes, and S. Spaccapietra. Object normal forms and dependency constraints for object-oriented schemata. *ACM ToDS*, 22:513–569, 1997.
39. B. Thalheim. *Dependencies in Relational Databases*. Teubner-Verlag, 1991.
40. B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag, 2000.
41. A. M. Tjoa and L. Berger. Transformation of requirement specifications expressed in natural language into an eer model. In *Entity-Relationship Approach*, volume 823. Springer Lecture Notes Series, 1993.
42. M. Vincent. *The semantic justification for normal forms in relational database design*. PhD thesis, Monash University, Melbourne, Australia, 1994.

oarieli@mta.ac.il

43. M. Vincent, J. Liu, and C. Liu. A redundancy free 4nf for xml. In *XML Database Symposium*, 2003.
44. W3C. Xml schema part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/#datatype, 2001.

oarieli@mta.ac.il

The Relative Complexity of Updates for a Class of Database Views

Stephen J. Hegner

Umeå University
Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Abstract. It is well known that the complexity of testing the correctness of an arbitrary update to a database view can be far greater than the complexity of testing a corresponding update to the main schema. However, views are generally managed according to some protocol which limits the admissible updates to a subset of all possible changes. The question thus arises as to whether there is a more tractable relationship between these two complexities in the presence of such a protocol. In this paper, this question is answered in the affirmative for closed update strategies, which are based upon the constant-complement approach of Bancilhon and Spyratos. Working within a very general framework which is independent of any particular data model, but which recaptures relational schemata constrained by so-called equality-generating dependencies (EGDs), (which include functional dependencies (FDs)), it is shown that the complexity of testing the correctness of a view update which follows a closed update strategy is no greater than that of testing a corresponding update to the main schema. In particular, if the main schema is relational and constrained by FDs, then there exists a set of FDs on the view, against which any candidate update may be tested for correctness. This holds even though the entire view may not be finitely axiomatizable, much less constrained by FDs alone.

1 Introduction

In a seminal work [1], Bancilhon and Spyratos showed how well-behaved update strategies for database views can be modelled in a very general framework using the so-called constant complement strategy. In more recent work, [2], [3], it is shown that by augmenting this basic framework with natural order structure, true uniqueness for so-called order-based updates may be obtained, in the sense that there is but one way to represent an update to the view in terms of an update to the main schema, regardless of the choice of complement. (*Order-based updates* are those which are realizable as a sequence of insertions and deletions.)

In this paper, the work of [2] and [3] is continued with an initial investigation of the complexity of determining whether a proposed view update is valid.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 155–175, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

This is hardly an idle question. Indeed, the axiomatization of a view may be infinitely more complex than that of the base schema, even in very simple cases. For example, the relational schema $\mathbf{E}_1$ with the single relation name $R[ABCD]$ on four attributes and the constraining set of functional dependencies (*FD*s) $\mathcal{F}_1 = \{A \to D, B \to D, CD \to A\}$, the projection view Π_{ABC} is not finitely axiomatizable [4].

Even without any special data structures, testing whether a relation satisfies a set of FDs is possible in time $O(n^2)$, with n the number of tuples in the relation, since it suffices to check each pair of tuples for conflict. If it is known that M is already a legal state and t is a tuple to be inserted, then testing whether $M \cup \{t\}$ satisfies the FDs may be performed in time $O(n)$. Under certain circumstances (e.g., with key dependencies), if the tuples are suitably indexed, these times may be reduced to $O(n)$ and $O(1)$, respectively. On the other hand, for the view Π_{ABC} identified above, neither test can be performed in worst-case $O(n^k)$ for any natural number k. Thus, the increase in complexity is indeed substantial, and certainly dashes any notion of tractability.

However, all is not lost, for testing an arbitrary proposed update to a view for correctness is far more general a task than is testing a proposed update under a closed update strategy. To address the latter idea, a notion of *relative complexity* is introduced, which takes into account that partial information regarding constraint satisfaction is already known about the proposed new view state. To illustrate, let $\mathbf{E}_2$ be the relational schema with the five-attribute relation $S[ABCDE]$, with constraints $\mathcal{F}_2 = \mathcal{F}_1 \cup \{A \to E\}$. The view to be updated is Π_{ABCE}, with the allowable updates those which hold the complementary view Π_{ABCD} constant. The updates which are allowed under the theory of closed update strategies are precisely those which hold the *meet* of these two views, Π_{ABC}, constant. Now Π_{ABCE} is not finitely axiomatizable, for the same reason that the view Π_{ABC} of $\mathbf{E}_1$ is not. However, since Π_{ABC} is to be held constant under any update to Π_{ABCE}, proposed updates need only be tested against the embedded FD $A \to E$; it is already known that the "ABC" part of the proposed new database is legal. Thus, the relative complexity of testing a proposed update to Π_{ABCE} is $O(n^2)$, the same as that for proposed updates to the main schema $\mathbf{E}_2$, *even though the view Π_{ABCE} itself is not finitely axiomatizable.* This idea is developed more formally in Example 4.15.

The main result of this paper is that this sort of result holds in a very general context; that is, if the complexity of testing the correctness of a potential database in the main schema is $O(n^k)$, then the relative complexity of testing the correctness of a potential database which is the result of a proposed update under a closed strategy is also $O(n^k)$.

A secondary result is also provided. In [2, 4.2] [3, 4.3], it is shown that the reflection to the main schema of an update to a closed view is unique, provided that the update is realizable as a sequence of legal insertions and deletions. In this paper, it is shown that the intermediate states in fact need not be legal; that it is enough that the update be realizable as sequence of insertions and deletions, and that the initial and final states be legal. In other words, essentially

oarieli@mta.ac.il

all updates under closed update strategies in an order-based framework reflect uniquely back to the main view. To illustrate, let $\mathbf{E}_3$ be the schema consisting of the single relation $R[ABC]$, constrained by $\mathcal{F}_3 = \{B \to A, B \to C\}$. The view to be updated is Π_{AB}, with Π_{BC} the complement to be held constant. The legal updates to Π_{AB}, none of which are insertions or deletions, are those which hold Π_B constant and which respect the FD $B \to A$; i.e., those which alter the A values of existing tuples. Thus, the results of [2] [3] do not apply. However, the extensions developed in Sect. 5 do, since the updates may be realized as sequences of insertions and deletions in which the intermediate states may violate the FD $B \to A$. A more detailed explanation is provided in Example 5.9.

2 An Overview of Existing Work

The results presented herein depend heavily upon the earlier work of the author on closed update strategies, which in turn depends upon the initial work of Bancilhon and Spyratos. To provide the reader with the essential background, this section contains two summaries. Summary 2.1 recaps the essential ideas of closed update strategies within the original set-based framework. Thus, it provides the essence of the framework of [1], although it is recast within the formalism of [2] [3]. Summary 2.2 sketches the key ideas developed in [2] [3] which are necessary to extend the set-based ideas to the order-based context.

While every effort has been made to keep this paper self contained, it may nevertheless be necessary to consult [2] and/or [3] to resolve detailed technical issues. Also, while the general theory is not attached to any particular data model, numerous examples are taken from the classical relational theory. Therefore, it is assumed that the reader is familiar with its standard notation and terminology.

Summary 2.1 (The classical results in the set-based framework). In the original work of Bancilhon and Spyratos [1], a database schema $\mathbf{D}$ is just a set. To maintain consistency with the more structured frameworks to be introduced shortly, this set will be denoted $\mathsf{LDB}(\mathbf{D})$ and called the *legal databases* of $\mathbf{D}$. Thus, a database schema is modelled by its instances alone; constraints, schema structure, and the like are not explicitly represented. A *morphism* $f : \mathbf{D}_1 \to \mathbf{D}_2$ of database schemata is a function $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$.

A *view* of the schema $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is a schema and $\gamma : \mathbf{D} \to \mathbf{V}$ is a surjective database morphism. A *morphism* $f : \Gamma_1 \to \Gamma_2$ of views $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ and $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ is a morphism $f : \mathbf{V}_1 \to \mathbf{V}_2$ of schemata such that the diagram to the right commutes. Following standard categorical terminology [5, 3.8], the morphism f is an *isomorphism* if there is a $g : \mathbf{V}_2 \to \mathbf{V}_1$ which is both a left and a right inverse to f. The *congruence* of Γ is the equivalence relation on $\mathsf{LDB}(\mathbf{D})$ defined by $(M_1, M_2) \in \mathsf{Congr}(\Gamma)$ iff $\gamma(M_1) = \gamma(M_2)$. It is easy to see that $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ and $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ of $\mathbf{D}$ are isomorphic iff $\mathsf{Congr}(\Gamma_1) = \mathsf{Congr}(\Gamma_2)$.

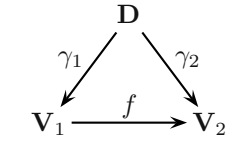


Fig. 1. View morphism

oarieli@mta.ac.il

An *update* on the schema $\mathbf{D}$ is a pair $(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$, which specifies a change of the state of $\mathbf{D}$ from M_1 to M_2. A *closed update family* U on $\mathbf{D}$ a set of updates which forms an equivalence relation; that is:

(up-r) For each $M \in \mathsf{LDB}(\mathbf{D})$, $(M, M) \in U$.
(up-s) If $(M_1, M_2) \in U$, then $(M_2, M_1) \in U$ as well.
(up-t) If $(M_1, M_2), (M_2, M_3) \in U$, then $(M_1, M_3) \in U$.

Thus, in a closed update family, all updates are reversible, and compatible updates are composable.

Now let $\Gamma = (\mathbf{V}, \gamma)$ be a view of the schema $\mathbf{D}$, and let U and T be closed update families for $\mathbf{D}$ and $\mathbf{V}$ respectively. An update strategy is a rule which translates updates on the view (i.e., in T) to updates on the main schema (i.e., in U). Formally, an *update strategy* for T with respect to U is a partial function $\rho : \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{V}) \to \mathsf{LDB}(\mathbf{D})$ which satisfies the following five conditions. (Here $X\downarrow$ means that X is defined.)

(upt:1) $\rho(M, N)\downarrow$ iff $(\gamma(M), N) \in T$. [Admissibility of an update depends only upon the state of $\mathbf{V}$, and not otherwise upon that of $\mathbf{D}$.]
(upt:2) If $\rho(M, N)\downarrow$, then $(M, \rho(M, N)) \in U$ and $\gamma(\rho(M, N)) = N$.
(upt:3) For every $M \in \mathsf{LDB}(\mathbf{D})$, $\rho(M, \gamma(M)) = M$. [Identity updates are reflected as identities.]
(upt:4) If $\rho(M, N)\downarrow$, then $\rho(\rho(M, N), \gamma(M)) = M$. [Every view update is globally reversible.]
(upt:5) If $\rho(M, N_1)\downarrow$ and $\rho(\rho(M, N_1), N_2)\downarrow$, then $\rho(M, N_2) = \rho(\rho(M, N_1), N_2)$. [View update reflection is transitive.]

The idea of such an update strategy is shown in Fig. 2 below. Put another way, $\rho : (\textsf{current state of } \mathbf{D}, \textsf{new state of } \mathbf{V}) \mapsto \textsf{new state of } \mathbf{D}$ in such a way that the new state of $\mathbf{D}$ gives rise to the desired new state of $\mathbf{V}$. The update to $\mathbf{V}$ must lie in T, and the reflected update to $\mathbf{D}$ must lie in U. In practice, U is often taken to be all possible updates on $\mathbf{D}$, i.e., $\mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$, but this is in no way essential to the theory.

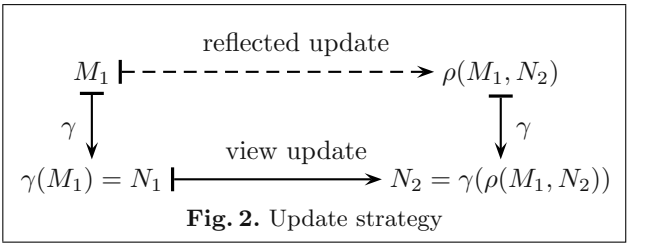


Fig. 2. Update strategy

Some authors have argued that closed update strategies are too restrictive to be of use [6]. However, as shown in [3], they are precisely the view updates which (a) do not depend upon the corresponding state of the main schema for admissibility, and (b) have their effect visible entirely within the view. In other words, they are the updates which can be understood entirely within the context of the view itself.

The idea that all view updates in a closed strategy have their effect contained entirely within the view itself is further manifested in their characterization via constant complement. A pair $\{\Gamma_1 = (\mathbf{V}_1, \gamma_1), \Gamma_2 = (\mathbf{V}_2, \gamma_2)\}$ of views of the schema $\mathbf{D}$ is called a subdirect complementary pair if it defines a lossless decomposition of $\mathbf{D}$. More precisely, the product $\Gamma_1 \times \Gamma_2 = (\mathbf{V}_{1\ \gamma_1 \otimes_{\gamma_2}} \mathbf{V}_2, \gamma_1 \otimes$

oarieli@mta.ac.il

γ_2) has $\mathsf{LDB}(\mathbf{V}_1 {}_{\gamma_1}\!\otimes_{\gamma_2}\mathbf{V}_2) = \{(\gamma_1(M), \gamma_2(M)) \mid M \in \mathsf{LDB}(\mathbf{D})\}$. The morphism $\gamma_1 \otimes \gamma_2 : \mathbf{D} \to \mathbf{V}_1 {}_{\gamma_1}\!\otimes_{\gamma_2}\mathbf{V}_2$ is given on elements by $M \mapsto (\gamma_1(M), \gamma_2(M))$. The pair $\{\Gamma_1, \Gamma_2\}$ forms a *subdirect complementary pair*, and Γ_1 and Γ_2 are called *subdirect complements* of one another, just in case $\gamma_1 \otimes \gamma_2$ is a bijection. In other words, $\{\Gamma_1, \Gamma_2\}$ is a subdirect complementary pair precisely in the case that the state of the schema $\mathbf{D}$ is recoverable from the combined states of $\mathbf{V}_1$ and $\mathbf{V}_2$.

If $\{\Gamma_1, \Gamma_2\}$ is a subdirect complementary pair, it is clear that there can be at most one update strategy on Γ_1 which holds Γ_2 constant. Specifically, define $\mathsf{UpdStr}\langle\Gamma_1, \Gamma_2\rangle : \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{V}_1) \to \mathsf{LDB}(\mathbf{D})$ by $(M, N) \mapsto (\gamma_1 \otimes \gamma_2)^{-1}(N, \gamma_2(M))$ whenever $(N, \gamma_2(M)) \in \mathsf{LDB}(\mathbf{V}_1 {}_{\gamma_1}\!\otimes_{\gamma_2}\mathbf{V}_2)$, and undefined otherwise. $\mathsf{UpdStr}\langle\Gamma_1, \Gamma_2\rangle$ is called the *update strategy* for Γ_1 with constant complement Γ_2. As first shown by Bancilhon and Spyratos [1, Thm. 7.3], every closed update strategy on a view Γ is of the form $\mathsf{UpdStr}\langle\Gamma, \Gamma'\rangle$ for some view Γ'. Specifically, let T and U closed update strategies for $\mathbf{V}$ and $\mathbf{D}$ respectively, and ρ an update strategy for T with respect to U. The *induced update family* on $\mathbf{D}$ is the smallest subset of U which will support the updates in T. It is denoted $\equiv_\rho$ and is given by $\{(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \mid (\exists N \in \mathsf{LDB}(\mathbf{V}))(\rho(M_1, N) = M_2)\}$. The *$\rho$-complement* of Γ, denoted $\widetilde{\Gamma}^\rho = (\widetilde{\mathbf{V}}^\rho, \widetilde{\gamma}^\rho)$, is defined to have $\mathsf{LDB}(\widetilde{\mathbf{V}}^\rho) = \mathsf{LDB}(\mathbf{D})/\equiv_\rho$, with the morphism $\widetilde{\gamma}^\rho : \mathbf{D} \to \widetilde{\mathbf{V}}^\rho$ given by $M \mapsto [M]_{\equiv_\rho}$; the latter denoting the equivalence class of M in $\equiv_\rho$. In other words, $(M_1, M_2) \in \mathsf{Congr}(\widetilde{\Gamma}^\rho)$ iff some view update $(N_1, N_2) \in T$ changes the state of $\mathbf{D}$ from $M_1 \in \gamma^{-1}(N_1)$ to $M_2 \in \gamma^{-1}(N_2)$. Thus, by construction, a potential update $(N_1, N_2) \in \mathsf{LDB}(\mathbf{V})$ is supported under ρ iff for some (resp. any) $M_1 \in \gamma^{-1}(N_1)$, there is an $M_2 \in \mathsf{LDB}(\mathbf{D})$ with $(M_1, M_2) \in \mathsf{Congr}(\widetilde{\Gamma}^\rho)$. In other words, the allowable updates to Γ under ρ are precisely those whose reflection into $\mathbf{D}$ leaves $\widetilde{\mathbf{V}}^\rho$ fixed; i.e., $\rho = \mathsf{UpdStr}\langle\Gamma, \widetilde{\Gamma}^\rho\rangle$.

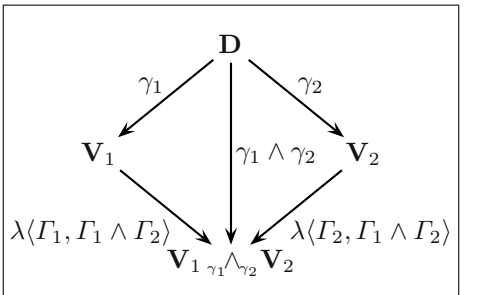


Fig. 3. Relative views

Not all subdirect complements give rise to closed update strategies. Condition (upt:1) mandates that the admissibility of a view update depend upon the state of the view alone. Thus, any information which is contained in the complement view and which is needed to determine the admissibility of an update must be contained in the view to updated as well. The necessary condition, first observed in [7, 2.10], is that the congruences must commute. Formally, the pair $\{\Gamma_1, \Gamma_2\}$ of views of $\mathbf{D}$ is called a *fully commuting pair* if $\mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2) = \mathsf{Congr}(\Gamma_2) \circ \mathsf{Congr}(\Gamma_1)$, with "$\circ$" denoting ordinary relational composition. A subdirect complementary pair $\{\Gamma_1, \Gamma_2\}$ which is fully commuting is called a *meet-complementary pair*, and Γ_1 and Γ_2 are called *meet complements* of one another. In this case, $\mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2)$ is also an equivalence relation on $\mathsf{LDB}(\mathbf{D})$, and so it is possible to define (up to isomorphism) the view $\Gamma_1 \wedge \Gamma_2 = (\mathbf{V}_1 {}_{\gamma_1}\!\wedge_{\gamma_2}\mathbf{V}_2, \gamma_1 \wedge \gamma_2)$ with $\mathsf{Congr}(\Gamma_1 \wedge \Gamma_2) = \mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2)$.

oarieli@mta.ac.il

The situation is summed up in the Fig. 3 above. Note that $\mathbf{V}_1 {}_{\gamma_1}\!\wedge_{\gamma_2} \mathbf{V}_2$ is a view not only of $\mathbf{D}$, but of $\mathbf{V}_1$ and $\mathbf{V}_2$ as well. Now define $\mathsf{UpdFam}\langle \Gamma_1, \Gamma_2 \rangle = \{(N_1, N_2) \in \mathsf{LDB}(\mathbf{V}_1) \times \mathsf{LDB}(\mathbf{V}_1) \mid \lambda\langle \Gamma_1, \Gamma_1 \wedge \Gamma_2 \rangle(N_1) = \lambda\langle \Gamma_1, \Gamma_1 \wedge \Gamma_2 \rangle(N_2)\}$. $\mathsf{UpdFam}\langle \Gamma_1, \Gamma_2 \rangle$ is a closed update family,, called the *update family* induced by Γ_2 on Γ_1. It embodies precisely the updates on Γ_1 which are admitted under $\mathsf{UpdStr}\langle \Gamma_1, \Gamma_2 \rangle$; in other words, the admissible updates to Γ_1 are precisely those which keep the meet $\Gamma_1 \wedge \Gamma_2$ constant. The key result [2, 3.9] [3, 3.10] is the following:

(a) For any update strategy ρ, $\mathsf{UpdStr}\langle \Gamma, \widetilde{\Gamma}^{\rho} \rangle = \rho$.
(b) For any meet complement Γ_1 of Γ, $\widetilde{\Gamma}^{\mathsf{UpdStr}\langle \Gamma, \Gamma_1 \rangle} = \Gamma_1$.

In the context of relational schema and views defined by projection, a pair of views forms a meet-complementary pair iff the decomposition is both lossless and dependency preserving [2, 2.16] [3, 2.17]. In this case, the meet view is just the projection on the common columns. To obtain an example in which the views form a subdirect complementary pair but not a meet complementary pair, it suffices to consider an example which is lossless but not dependency preserving.

In [8], the connection between decompositions of database schemata and commuting congruences is investigated thoroughly.

Summary 2.2 (The order-based framework). Despite its simplicity and elegance, the set-based framework for closed update strategies has a substantial shortcoming; namely, the update strategy depends upon the choice of the complement. For example, let $\mathbf{E}_1$ be the relational schema with the single relation $R[ABC]$, constrained by the single FD $B \to C$, and let Π_{AB} be the view defined by the projection mapping π_{AB}. Define Π_{BC} similarly. Since the pair $\{\Pi_{AB}, \Pi_{BC}\}$ forms a lossless and dependency-preserving decomposition of $\mathbf{E}_1$, it also forms a meet-complementary pair [2, 2.16] [3, 2.17]. Indeed, Π_{BC} is the "natural" complement of Π_{AB}, and the one which yields the "obvious" strategy for reflecting updates to Π_{AB} back to $\mathbf{F}_0$. However, as shown in [2, 1.3] [3, 1.3], it is possible to find other complements of Π_{AB} which have exactly the same meet, and so support exactly the same updates to Π_{AB}. Although these alternate complements are a bit pathological, the set-based theory outlined above in Summary 2.1 does not prefer Π_{BC} to them in any way.

To formalize this preference, additional structure must be incorporated into the model. Most database models incorporate some sort of order structure. In the relational model, the databases may be ordered via relation-by-relation inclusion. Furthermore, the common database mappings built from projection, restriction, and join are all order preserving with respect to this natural order structure. In particular, while the views Π_{AB} and Π_{BC} are order mappings, the alternate views identified in [2, 2.16] [3, 2.17] are not.

The theory developed in [2] and [3] provides a systematic extension to the results outlined in Summary 2.1 above to the order-based setting. A *order schema* $\mathbf{D}$ is taken to be a partially ordered set (*poset*) $(\mathsf{LDB}(\mathbf{D}), \leq_{\mathbf{D}})$. A *order database mapping* $f : \mathbf{D}_1 \to \mathbf{D}_2$ is an order-preserving function; i.e., $M_1 \leq_{\mathbf{D}_1} M_2$ implies $f(M_1) \leq_{\mathbf{D}_2} f(M_2)$. An *order view* $\Gamma = (\mathbf{V}, \gamma)$ of $\mathbf{D}$ consists of an order schema $\mathbf{V}$ and an open surjection $\gamma : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V})$; that is, a surjection which

oarieli@mta.ac.il

is order preserving and, in addition, which satisfies the property that whenever $N_1 \leq_{\mathbf{V}} N_2$, there are $M_1, M_2 \in \mathsf{LDB}(\mathbf{D})$ with the properties that $M_1 \leq_{\mathbf{D}} M_2$, $f(M_1) = N_1$, and $f(M_2) = N_2$. For the pair of order views $\{\Gamma_1, \Gamma_2\}$ to be a subdirect complementary pair in the order sense, the mapping $\gamma_1 \otimes \gamma_2 : \mathbf{D} \to \mathbf{V}_1 \,{}_{\gamma_1}\!\otimes_{\gamma_2} \mathbf{V}_2$ must be an order isomorphism, and not merely an order-preserving bijection. To obtain a closed update strategy in the oder-bases sense, conditions (upt:1)-(upt:5) identified in Summary 2.1 are augmented with the following three additions.

(upt:6) If $\rho(M, N)\downarrow$ and $\gamma(M) \leq_{\mathbf{V}} N$, then $M \leq_{\mathbf{D}} \rho(M, N)$. [View update reflects order.]

(upt:7) If $\rho(M_1, N_1)\downarrow$ with $M_1 \leq_{\mathbf{D}} \rho(M_1, N_1)$, then for all $M_2 \in \mathsf{LDB}(\mathbf{D})$ with $M_1 \leq_{\mathbf{D}} M_2 \leq_{\mathbf{D}} \rho(M_1, N_1)$, there is an $N_2 \in \mathsf{LDB}(\mathbf{V})$ with $\rho(M_1, N_2) = M_2$ and $\rho(M_2, \gamma(\rho(M_1, N_1))) = \rho(M_1, N_1)$. [This condition is called *order completeness*.]

(upt:8) If $M_1, M_2 \in \mathsf{LDB}(\mathbf{D})$ with $M_1 \leq_{\mathbf{D}} M_2$, then for every $N_1, N_2 \in \mathsf{LDB}(\mathbf{V})$ for which $N_1 \leq_{\mathbf{V}} N_2$, $\rho(M_1, N_1)\downarrow$, and $\rho(M_2, N_2)\downarrow$, if must be the case that $\rho(M_1, N_1) \leq_{\mathbf{D}} \rho(M_2, N_2)$. [This condition is called *order reflection*.]

Modulo these modification, it is fair to say, at least in a general way, that [2] and [3] extend the classical set-based constant complement theory to the order-based setting. Within the setting of this extension, a number of uniqueness results are obtained. Most importantly, while complements are not necessarily unique, order-based updates are. Specifically, let $\mathbf{D}$ be a database schema, and let U be a closed update family for $\mathbf{D}$. A pair $(M_1, M_2) \in U$ is called: a *formal insertion* with respect to U if $M_1 \leq_{\mathbf{D}} M_2$; a *formal deletion* with respect to U if $M_2 \leq_{\mathbf{D}} M_1$; and an *order-based update* with respect to U if it is a composition of a sequence of formal insertions and formal deletions. The main theorem [2, 4.2] [3, 4.3] states that for an order-based view $\Gamma = (\mathbf{V}, \gamma)$ of the order-based schema $\mathbf{D}$, all order-based closed update strategies must agree on all order-based updates. In other words, there is only one way to reflect the view update back to the main schema. This does not depend upon the choice of complement, or even the value of the meet. It is unique, period. As a rich source of classical but important examples, all SPJR-mappings (Select, Project, Join, Rename) in the classical relational setting define order views [2, 2.5] [3, 2.5].

In [9], a theory of *direct* decomposition (i.e., situations in which the views are independent and so the meet is trivial) of order-based schemata is presented.

3 A Framework for Modelling View Updates

The framework described in Summary 2.2 must be extended in two essential ways in order to recapture the key ideas involved in updates and their complexity. First of all, to recapture complexity, it must be possible to characterize the size of a database, and also the size of an update. Secondly, to recapture admissibility of a candidate database, it must be possible to discuss both those databases which satisfy the underlying constraints and those which do not. Fortunately, there is a very simple model which meets both of these requirements. To begin, the underlying ideas in the world of posets are developed.

oarieli@mta.ac.il

Definition 3.1 (CFA-posets and morphisms). Let X be any set (not necessarily finite), and let $\mathscr{P}_{\mathsf{f}}(X) = (\mathcal{P}_{\mathsf{f}}(X), \subseteq)$ denote the poset consisting of all finite subsets of X, ordered under set inclusion. A *concrete finitely-atomistic poset* $\mathbf{P} = (P, \subseteq)$ *(over* X*)* (*CFA-poset* for short) is any sub-poset of $\mathscr{P}_{\mathsf{f}}(X)$ which contains the least element $\emptyset$, as well as all singletons of the form $\{x\}$ with $x \in X$. Define $\mathsf{Atoms}(\mathbf{P}) = \{\{x\} \mid x \in X\}$; these are clearly the atoms of this poset in the abstract sense [10, 5.2]. The *basis* of any $p \in P$ is $\mathsf{Basis}_{\mathbf{P}}(p) = \{a \in \mathsf{Atoms}(\mathbf{P}) \mid a \subseteq p\} = \{\{x\} \mid x \in p\}$. The term *finitely atomistic* is borrowed from the lattice-theoretic world [11, p. 234], and refers to the fact that every element in $\mathbf{P}$ is the supremum of the atoms which are less than it; i.e., $p = \sup\{a \in \mathsf{Atoms}(\mathbf{P}) \mid a \subseteq p\}$. Note also that X may be recovered from $\mathbf{P}$; define $\mathsf{Foundation}(\mathbf{P}) = \bigcup \mathsf{Atoms}(\mathbf{P}) = \{x \mid \{x\} \in \mathsf{Atoms}(\mathbf{P})\}$. Thus, it is safe to speak of a CFA-poset without explicitly identifying the underlying set.

To avoid confusion when more than one poset is considered, $\bot_{\mathbf{P}}$ will be used to denote $\emptyset$ when it is regarded as the least element of $\mathbf{P}$. Finally, it is often useful to have a notation for atoms and basis when the least element is included as well; thus $\mathsf{ExtAtoms}(\mathbf{P}) = \mathsf{Atoms}(\mathbf{P}) \cup \{\bot_{\mathbf{P}}\}$, and for $p \in P$, $\mathsf{ExtBasis}_{\mathbf{P}}(p) = \mathsf{Basis}_{\mathbf{P}}(p) \cup \{\bot_{\mathbf{P}}\}$,

Let $\mathbf{P} = (\mathsf{LDB}(\mathbf{P}), \subseteq)$ and $\mathbf{Q} = (\mathsf{LDB}(\mathbf{Q}), \subseteq)$ be CFA-posets. A *CFA-morphism* is a function $f : P \to Q$ with the property that it is *basis preserving*, in the precise sense that for all $p \in P$, $\bigcup\{f(a) \mid a \in \mathsf{Basis}_{\mathbf{P}}(p)\} = \mathsf{Basis}_{\mathbf{Q}}(f(p))$. It is clear that a CFA-morphism is *monotone*, i.e., $p_1 \subseteq p_2$ implies $f(p_1) \subseteq f(p_2)$, and thus a poset morphism in the ordinary sense. Furthermore, $f(\bot_{\mathbf{P}}) = \bot_{\mathbf{Q}}$, since $\mathsf{Basis}_{\mathbf{P}}(\bot_{\mathbf{P}}) = \mathsf{Basis}_{\mathbf{Q}}(\bot_{\mathbf{Q}}) = \emptyset$. Thus, the behavior of a basis-preserving morphism is determined entirely by its action on the atoms of the poset.

A CFA-morphism which is surjective is called a *CFA-surjection*. The CFA-surjection $f : \mathbf{P} \to \mathbf{Q}$ is *open* if, for each pair $q_1, q_2 \in Q$ with $q_1 \subseteq q_2$, there are $p_1, p_2 \in P$ with the properties that $p_1 \subseteq p_2$, $f(p_1) = q_1$, and $f(p_2) = q_2$. If f is an open surjection, then $\mathbf{Q}$ carries the weakest order which renders f monotone.

Definition 3.2 (CFA-schemata, morphisms, and views). Formally, a *concrete finitely atomistic database schema* (*CFA-schema* for short) $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ is just a CFA-poset. A *CFA-database morphism* is just a CFA-morphism in the sense given above. A *CFA-view* is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is a CFA-schema and γ is an open CFA-surjection.

Example 3.3 (Relational CFA-schemata, morphisms, and views). Let $\mathbf{R}$ be a relational schema consisting of a single relation $R[\mathbf{A}]$, with a family $\mathcal{C}$ of constraints; $\mathsf{LDB}(\mathbf{R})$ the set of all finite relations satisfying those constraints. $\mathbf{R}$ is automatically an order schema in the sense of [2] and [3], with the order defined by set inclusion. For it to be a CFA-schema, it must also be finitely atomistic. Specifically, this means that both the empty relation $\emptyset$ and each set $\{t\}$ containing exactly one tuple satisfies the constraints of $\mathcal{C}$. These conditions are satisfied, for example, whenever $\mathcal{C}$ consists of universal dependencies, such as *full dependencies* [12, Ch. 10].) These sets are very broad, and include *equality generating dependencies* (EGDs) such as FDs, and *tuple generating dependencies*

oarieli@mta.ac.il

(TGDs) such as join dependencies. They do not, however, include dependencies involving existential quantification, such as *inclusion dependencies* [12, Ch. 9], upon which foreign-key dependencies are based. A similar construction applies in the case in which $\mathbf{R}$ contains several relations; in this case, the atoms are those instances in which one relation contains one tuple, and the rest are empty.

Both projection and restriction are examples of open CFA-surjections in the relational context, since each is defined by its action on single tuples. Thus, they define CFA-views. On the other hand, joins are not in general CFA-morphisms, since they are not basis preserving.

Definition 3.4 (Unconstrained databases). While the legal databases consist of some finite subsets of the foundation, the potential databases consist of all of them. Specifically, Let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, and define $(\overline{\mathbf{D}} = \mathsf{DB}(\mathbf{D}), \subseteq)$ with $\mathsf{DB}(\mathbf{D}) = \mathscr{P}_{\mathsf{f}}(\mathsf{Foundation}(\mathbf{D}))$, and let $\iota_{\mathbf{D}} : \mathbf{D} \to \overline{\mathbf{D}}$ be the identity embedding; $M \mapsto M$. The elements of $\mathsf{DB}(\mathbf{D})$ are called the *unconstrained databases* of $\mathbf{D}$. In the relational example $\mathbf{R}$ identified in Example 3.3, $\mathsf{DB}(\mathbf{D})$ is the set of all finite relations on attribute set $\mathbf{A}$, regardless of whether or not the constraints of $\mathcal{C}$ are met.

Remark 3.5 (Abstract FA-schemata). A cornerstone of the framework developed in [2] [3] is that equivalence up to poset isomorphism is adequate to characterize a database schema. In other words, the theory is indifferent to such "inessential" variations. However, in the approach taken here, this is not the case. Rather, the database schemata are required to have a specific form; namely, that of a sub-poset of a power set. This is not an essential change. It is quite possible to develop the theory of this paper along the lines of *abstract FA-posets*, which may be axiomatized independently of any reference to CFA-posets, but which amount to those posets which are isomorphic to CFA-posets. The reason for not taking this direction is that it becomes much less intuitive, and much more cumbersome notationally, to define the unconstrained databases. The gains realized in having such a natural model for going from constrained to unconstrained databases seems worth the loss in abstraction. The more concrete approach is not entirely without its drawbacks, however. In particular, views which are constructed axiomatically must then be "concretized." For example, at the end of Definition 4.3, a method for constructing a CFA-view from a congruence is provided.

Definition 3.6 (k-models and subinstance properties). In a relational schema $\mathbf{R}$ constrained by a family $\mathcal{F}$ of FDs, to test a candidate relation M for legality, it suffices to test each pair of tuples for conflict. In other words, if every two-element subset of M satisfies $\mathcal{F}$, then M itself does. For more general families of EGDs, a corresponding property requiring the testing of k tuples at a time is easily formulated. The following notions extend these ideas to the abstract framework.

Let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-database schema, and let $k \in \mathbb{N}$. A *k-model* of $\mathbf{D}$ is any $M \in \mathsf{DB}(\mathbf{D})$ with the property that every $N \in \mathsf{DB}(\mathbf{D})$ with $N \subseteq M$

oarieli@mta.ac.il

and $\mathsf{Card}(N) \leq k$ is in $\mathsf{LDB}(\mathbf{D})$. (Here $\mathsf{Card}(X)$ denotes the cardinality of the set X.) The schema $\mathbf{D}$ has the *k-submodel property* if, for every $M \in \mathsf{DB}(\mathbf{D})$, $M \in \mathsf{LDB}(\mathbf{D})$ iff it is a k-model of $\mathbf{D}$. The schema $\mathbf{D}$ is *closed under subinstances* if, for any $M \in \mathsf{LDB}(\mathbf{D})$ and any $N \subseteq M$, $N \in \mathsf{LDB}(\mathbf{D})$ as well. The following observation is immediate.

Observation 3.7. *If $\mathbf{D}$ is a CFA-schema which has the k-submodel property for some natural number k, then it is closed under subinstances.* □

Remark 3.8 (The limits of k-models). While k-models recapture the idea of FDs in particular and EGDs in general, they do not recapture the tuple-generating properties of join dependencies in particular and TGDs in general. For such dependencies, a more general notion, the (k_1, k_2)-model, is needed. See the comments in Discussion 6.1.

Remark 3.9 (The measure of complexity). In a simple sense, a potential database $M \in \mathsf{DB}(\mathbf{D})$ of a schema $\mathbf{D}$ with the k-submodel property may be tested for membership in $\mathsf{LDB}(\mathbf{D})$ in worst-case time $O(n^k)$, with $n = \mathsf{Card}(M)$, since there are $\binom{n}{k}$ subsets of size k to test, and $O(\binom{n}{k}) = O(n^k)$, when k is taken to be constant and n the variable. In the context of simple update operations, this complexity may be further reduced. For example, if the update corresponds to the insertion of a single atom, then the complexity is $O(n^{k-1})$, since the only k-element subsets which need be checked are those which contain the new atom. In view of Observation 3.7, no checking is needed at all for deletions.

In certain contexts, with the support of appropriate data structures, these values may be reduced even further. Most notably, with key dependencies in the case of FDs, satisfaction may be performed in linear time, and the correctness of simple insertions may be determined in constant time [13]. For reasons of this dependence upon data structures, as well as space constraints, these issues will not be pursued further in this paper. Rather, complexity will be characterized solely in terms of k-submodel properties.

To close this section, a few essential properties of schemata which are closed under subinstances are developed.

Definition 3.10 (Strong morphisms and injective generators). Let $\mathbf{D}_1 = (\mathsf{LDB}(\mathbf{D}_1), \subseteq)$ and $\mathbf{D}_2 = (\mathsf{LDB}(\mathbf{D}_2), \subseteq)$ be CFA-schemata, and let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a CFA-surjection.

(a) The morphism f is a *downwardly strong* if for every $M_1 \in \mathsf{LDB}(\mathbf{D}_1)$ and $M_2 \in \mathsf{LDB}(\mathbf{D}_2)$ for which $M_2 \subseteq f(M_1)$, there is an $M_1' \in \mathsf{LDB}(\mathbf{D}_1)$ with $M_1' \subseteq M_1$ and $f(M_1') = M_2$. Note in particular that if f is surjective and downwardly strong, then it must be open as well.

(b) The morphism f is *injectively generating* if for every $M_2 \in \mathsf{LDB}(\mathbf{D}_2)$, there is an $M_1 \in f^{-1}(M_2)$ with the property that $\mathsf{Card}(M_1) = \mathsf{Card}(M_2)$.

Proposition 3.11. *Let $\mathbf{D}_1 = (\mathsf{LDB}(\mathbf{D}_1), \subseteq)$ and $\mathbf{D}_2 = (\mathsf{LDB}(\mathbf{D}_2), \subseteq)$ be CFA-database schemata, with $f : \mathbf{D}_1 \to \mathbf{D}_2$ a CFA-surjection and $\mathbf{D}_1$ closed under subinstances. Then f is downwardly strong and injectively generating, and $\mathbf{D}_2$ is closed under subinstances.*

oarieli@mta.ac.il

Proof. To show that f is downwardly strong, let $M \in \mathsf{LDB}(\mathbf{D}_1)$ and $N \in \mathsf{LDB}(\mathbf{D}_2)$ with $f(M) = N$, and let $N' \in \mathsf{LDB}(\mathbf{D}_2)$ with $N' \subseteq N$. Set $M' = \{A \in M \mid f(A) \in N'\}$. Then $M' \in \mathsf{LDB}(\mathbf{D}_1)$ (since $\mathbf{D}_1$ is closed under subinstances) with $f(M') = N'$.

To show that f is injectively generating, let $N \in \mathsf{LDB}(\mathbf{D}_2)$ and choose $M \in f^{-1}(N)$. For each $B \in N$, choose exactly one $A_B \in M$ with the property that $f(A_B) = B$, and put $M' = \{A_B \mid B \in N\}$. Since $\mathbf{D}_1$ is closed under subinstances, $M' \in \mathsf{LDB}(\mathbf{D}_1)$, with $f(M') = N'$.

Finally, to show that $\mathbf{D}_2$ is closed under subinstances, let $N \in \mathsf{LDB}(\mathbf{D}_2)$ and let $N' \in \mathsf{DB}(\mathbf{D})$ with $N' \subseteq N$. Choose $M \in f^{-1}(N)$ and set $M' = \{A \in M \mid f(A) \in N'\}$. Since $\mathbf{D}_1$ is closed under subinstances, $M' \in \mathsf{LDB}(\mathbf{D}_1)$, and so $N' \in \mathsf{LDB}(\mathbf{D}_2)$ with $f(M') = N'$. □

4 View Constructions and Relative Complexity

The ultimate goal of this section is the proof of the main theorem of this paper — that the relative complexity of view update for a closed view is no greater than that of update in the base schema. To achieve this result, certain key results from [2] [3] must be lifted to the current, more structured framework.

To begin, it is shown that a functional connection, or, equivalently, a subsumption of congruences is sufficient to define a relative view.

Lemma 4.1 (CFA-view fill-in). *Let* $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ *be a CFA-schema, let* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ *be CFA-views of* $\mathbf{D}$*, and let* $f : \mathbf{V}_1 \to \mathbf{V}_2$ *be any function which renders the diagram of Fig. 1 commutative. Then* f *is necessarily an open CFA-surjection, and hence defines a relative CFA-view* $\Lambda(\Gamma_1, \Gamma_2) = (\mathbf{V}_2, \lambda\langle\Gamma_1, \Gamma_2\rangle)$ *with* $\lambda\langle\Gamma_1, \Gamma_2\rangle = f$.

Proof. It is immediate that f is surjective, since γ_2 is. To show that it is open, let $M, N \in \mathsf{LDB}(\mathbf{V}_2)$ with $M \subseteq N$. Then, since γ_2 is open, there are $M', N' \in \mathsf{LDB}(\mathbf{D})$ with $M' \subseteq N'$ and $f(M') = M$, $f(N') = N$. Then $\gamma_1(M') \subseteq \gamma_1(N')$ with $\gamma_1(M') \in f^{-1}(M)$, $\gamma_1(N') \in f^{-1}(N)$; i.e., f is open. Finally, to show that it is basis preserving, let $N \in \mathsf{LDB}(\mathbf{V}_1)$; then, since γ_1 is surjective, there exists $M \in \mathsf{LDB}(\mathbf{D})$ with $\gamma_1(M) = N$. Since γ_1 is basis preserving, each $a \in N$ is of the form $\gamma_1(b)$ for some $b \in M$. Thus $f(N) = f(\gamma_1(M)) = \gamma_2(M) = \{\gamma_2(b) \mid b \in M\} = \{f(\gamma_1(b)) \mid b \in M\} = \{f(a) \mid a \in N\}$. □

Proposition 4.2. *Let* $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ *be a CFA-schema and let* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ *be CFA-views of* $\mathbf{D}$*. Then there is a view morphism* $f : \Gamma_1 \to \Gamma_2$ *iff* $\mathsf{Congr}(\Gamma_1) \subseteq \mathsf{Congr}(\Gamma_2)$. □

In the order-based context, the congruences which define order views are the *order-compatible congruences* [2, 2.9] [3, 2.9]. In the present framework, the appropriate condition is that of being *atomically generated*; that is, of being defined entirely by the behavior on the basis of the underlying schema. The formal details are as follows.

oarieli@mta.ac.il

Definition 4.3 (Atomically generated equivalence relations). Let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, and let R be an equivalence relation on $\mathsf{LDB}(\mathbf{D})$. Define $[\mathbf{D}]_R = (\mathsf{LDB}(\mathbf{D})/R, \leq_{[\mathbf{D}]_\mathbf{R}})$, with $\mathsf{LDB}(\mathbf{D})/R$ the set of equivalence classes of R, and $\leq_{[\mathbf{D}]_\mathbf{R}}$ the relation on $\mathsf{LDB}(\mathbf{D})/R$ given by $[M]_R \leq_{[\mathbf{D}]_\mathbf{R}} [N]_R$ iff $(\exists M_1 \in [M]_R)(\exists N_1 \in [N]_R)(M_1 \leq_\mathbf{D} N_1)$, and define $\theta_R : \mathbf{D} \to \mathsf{LDB}(\mathbf{D})/R$ by $M \mapsto [M]_R$. In the order-based setting, it is already known that $[\mathbf{D}]_R$ is an order schema, and that $\Theta_R = ([\mathbf{D}]_R, \theta_R)$ is an order-based view [2, 2.9] [3, 2.9].

To extend this result to the CFA-setting, a few additional conditions must be imposed. The idea is that since CFA-morphisms are completely characterized by their action on atoms, the corresponding notion of equivalence relation must have this same property. Specifically, let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, and let R be an equivalence relation on $\mathsf{LDB}(\mathbf{D})$. Informally, R is atomically generated if its equivalence classes are defined entirely by the equivalences of its atoms. Formally, R is *atomically generated* if, for any $(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$, $(M_1, M_2) \in R$ iff the following two conditions are satisfied.

(atg:1) $(\forall A_1 \in \mathsf{Basis}_\mathbf{D}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_\mathbf{D}(M_2))((A_1, A_2) \in R))$

(atg:2) $(\forall A_2 \in \mathsf{Basis}_\mathbf{D}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_\mathbf{D}(M_1))((A_1, A_2) \in R))$

Put another way, define the *atomic subequivalence* $\mathsf{AtomicEq}(R) = R \cap (\mathsf{ExtAtoms}(\mathbf{D}) \times \mathsf{ExtAtoms}(\mathbf{D}))$. Then, for R to be atomically generated, it must be entirely recoverable from $\mathsf{AtomicEq}(R)$.

It is easy to see that an atomically generated equivalence relation provides the correct construction for obtaining an abstract FA-view; however, such a view is not concrete because the equivalence class $[M]_R$ is not the union of its basis. Nonetheless, this is easy to fix. For any $M \in \mathsf{LDB}(\mathbf{D})$, let $[\![M]\!]_R = \{[x]_R \mid x \in M\}$, let $\mathsf{LDB}(\mathbf{D})/\!\!/R = \{[\![M]\!]_R \mid M \in \mathsf{LDB}(\mathbf{D})\}$, and let $[\![\mathbf{D}]\!]_R = (\mathsf{LDB}(\mathbf{D})/\!\!/R, \subseteq)$. Define $[\![\Theta_R]\!] = ([\![\mathbf{D}]\!]_R, [\![\theta_R]\!])$, with $[\![\theta_R]\!] : \mathbf{D} \to [\![\mathbf{D}]\!]_R$ given by $M \mapsto [\![M]\!]_R$. This construction provides a CFA-view whose congruence is R; Lemma 4.4 and Proposition 4.5 below formalize this fact.

Lemma 4.4 (Concretization of views defined by equivalence relations). *Let* $\mathbf{D}$ *be a CFA-schema, and let* R *be an atomically generated equivalence relation on* $\mathsf{LDB}(\mathbf{D})$. *Then* $[\![\mathbf{D}]\!]_R$ *is a CFA-schema with* $\mathsf{Atoms}([\![\mathbf{D}]\!]_\mathbf{R}) = \{[\![a]\!]_R \mid a \in \mathsf{Atoms}(\mathbf{D})\}$, *and* $[\![\Theta_R]\!] = ([\![\mathbf{D}]\!]_R, [\![\theta_R]\!])$ *is a CFA-view of* $\mathbf{D}$ *with* $\mathsf{Congr}([\![\Theta_R]\!]) = R$. □

Proposition 4.5 (Characterization of CFA-views). *Let* $\mathbf{D}$ *be a CFA-schema, and let* R *be any equivalence relation on* $\mathsf{LDB}(\mathbf{D})$. *Then* $[\![\Theta_R]\!]$ *is a CFA-view iff* R *is an atomically generated equivalence. In particular, if* $\Gamma = (\mathbf{V}, \gamma)$ *is a CFA-view, then* $\mathsf{Congr}(\Gamma)$ *is an atomically generated equivalence.* □

A critical component of the theory is the ability to "lift" the constructions on a constrained schema $\mathbf{D}$ to the associated unconstrained schema $\overline{\mathbf{D}}$. This includes also morphisms between such schemata, views, and even equivalence relations induced by views. To begin, the notion of lifting a morphism is introduced.

oarieli@mta.ac.il

Definition 4.6 (Extensions of CFA-morphisms to unconstrained databases). Given CFA-schemata $\mathbf{D}_1 = (\mathsf{LDB}(\mathbf{D}_1), \subseteq)$ and $\mathbf{D}_2 = (\mathsf{LDB}(\mathbf{D}_2), \subseteq)$ and a CFA-morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$, there is a unique natural extension $\overline{f} : \overline{\mathbf{D}}_1 \to \overline{\mathbf{D}}_2$ which renders the diagram of Fig. 4 to the right commutative. Specifically, for any $M \in \mathsf{DB}(\mathbf{D}_1)$, define $\overline{f}(M) = \bigcup\{f(\{x\}) \mid x \in M\}$. The following is easy to verify.

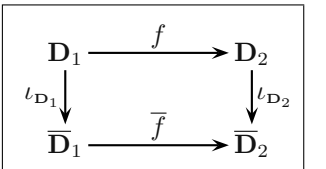


Fig. 4. The extension of a CFA-morphism

Proposition 4.7. *Let* $\mathbf{D}_1 = (\mathsf{LDB}(\mathbf{D}_1), \subseteq)$ *and* $\mathbf{D}_2 = (\mathsf{LDB}(\mathbf{D}_2), \subseteq)$ *be CFA-schemata, and let* $f : \mathbf{D}_1 \to \mathbf{D}_2$ *be a CFA-morphism. Then* $\overline{f} : \overline{\mathbf{D}}_1 \to \overline{\mathbf{D}}_2$ *is also a CFA-morphism, and it is an (open) CFA-surjection iff* f *is.* □

Discussion 4.8 (Lifting of entire diagrams and completions of CFA-equivalences). The lifting construction described in Definition 4.6 may be applied to any commutative diagram containing CFA-morphisms. Thus, the commutative diagram shown in Fig. 5 to the right may be obtained from the commutative diagram shown in Fig. 3. The key results of this section, however, require a further step; namely, that the extension operation be moved from an entire construction to the individual components.

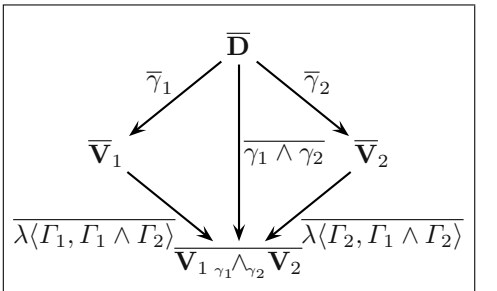


Fig. 5. Entire diagram lifting

In concrete terms, it is necessary to show that the diagram of Fig. 5 is the same, component-by-component and morphism-by-morphism, as that of Fig. 6 to the right. The key to establishing this equivalence lies within the associated equivalence relations. Specifically, given a CFA-schema $\mathbf{D}$, it is the case that the equivalence relation of $\overline{\mathbf{D}}$ is a natural completion of that of $\mathbf{D}$. More formally, proceed as follows. Let R be a CFA-equivalence on the CFA-schema $\mathbf{D}$. Define the *completion* of R to be the equivalence relation $\overline{R}$ on $\overline{\mathbf{D}}$ with the property that for $M_1, M_2 \in \mathsf{DB}(\mathbf{D})$,

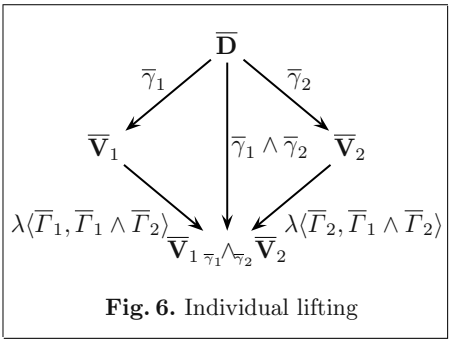


Fig. 6. Individual lifting

$$(M_1, M_2) \in \overline{R} \Leftrightarrow (((\forall A_1 \in \mathsf{Basis}_{\mathbf{D}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\mathbf{D}}(M_2))((A_1, A_2) \in R)) \wedge ((\forall A_2 \in \mathsf{Basis}_{\mathbf{D}}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\mathbf{D}}(M_1))((A_1, A_2) \in R)))$$

oarieli@mta.ac.il

Definition 4.9 (Independence dependencies). Consider a simple relational schema $R[ABC]$ constrained by the single FD $B \to C$, which decomposes losslessly and in a dependency-preserving fashion into the two projections Π_{AB} and Π_{BC}. There are two equivalent ways of identifying the view states which may be combined to form a state of the main schema. First, one may say that the projection of each view on attribute B is the same. Second, on may say that for each tuple (a, b) of the view Π_{AB}, there is a tuple (b, c) with a matching B-value, and conversely. These conditions are so obviously identical that it may seem pointless to differentiate between them. However, in a more general context, they display an important difference. The first (characterization by equivalent projections) does not make use explicit use of individual tuples, and thus claims a generalization as the *Π_B-independence dependency* within the framework of [2] [3]. On the other hand, a generalization of the second condition (tuple-by-tuple matching) requires a corresponding abstraction of the notion of a tuple; while the framework of [2] [3] does not admit such an abstraction, the more structured one used here does. The formalizations are as follows.

Let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, let $\{\Gamma_1, \Gamma_2\}$ be a subdirect complementary pair of CFA-views, and let $\Gamma_3 = (\mathbf{V}_3, \gamma_3)$ be a view of $\mathbf{D}$, with $\mathsf{Congr}(\Gamma_1) \subseteq \mathsf{Congr}(\Gamma_3)$ and $\mathsf{Congr}(\Gamma_2) \subseteq \mathsf{Congr}(\Gamma_3)$. The *$\Gamma_3$-independence dependency* on $\mathbf{V}_1 \,{}_{\gamma_1}\!\otimes_{\gamma_2} \mathbf{V}_2$, denoted $\otimes_{\Gamma_3}$, is satisfied iff for any $M_1 \in \mathsf{LDB}(\mathbf{V}_1)$ and $M_2 \in \mathsf{LDB}(\mathbf{V}_2)$, the following condition is satisfied ([2, 2.12], [3, 2.13]).

(id) $\quad ((M_1, M_2) \in \mathsf{LDB}(\mathbf{V}_1 \,{}_{\gamma_1}\!\otimes_{\gamma_2} \mathbf{V}_2)) \Leftrightarrow (\lambda\langle\Gamma_1, \Gamma_3\rangle(M_1) = \lambda\langle\Gamma_2, \Gamma_3\rangle(M_2))$

On the other hand, the *pointwise Γ_3-independence dependency* is satisfied iff the following two dual conditions are met.

(id:1)
$(\forall A_1 \in \mathsf{Basis}_{\mathbf{V}_1}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\mathbf{V}_2}(M_2))(\lambda\langle\Gamma_1, \Gamma_3\rangle(A_1) = \lambda\langle\Gamma_2, \Gamma_3\rangle(A_2))$

(id:2)
$(\forall A_2 \in \mathsf{Basis}_{\mathbf{V}_1}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\mathbf{V}_2}(M_1))(\lambda\langle\Gamma_1, \Gamma_3\rangle(A_1) = \lambda\langle\Gamma_2, \Gamma_3\rangle(A_2))$

In the context of CFA-views, conditions (id:1) and (id:2) may replace (id).

Proposition 4.10. *Let $(\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, and let $\{\Gamma_1, \Gamma_2\}$ be a subdirect complementary pair of CFA-views of $\mathbf{D}$. Then it is also a meet complementary pair iff conditions (id:1) and (id:2) of Definition 4.9 are satisfied.*

Proof. Follows directly from the discussion of Definition 4.9 and [2, 2.13] [3, 2.14]. □

Lemma 4.11 (Commuting congruences for completions). *Let $\mathbf{D}$ be a CFA-schema and let $M_1, M_2, \in \mathsf{DB}(\mathbf{D})$.*

(a) *Let $\Gamma = (\mathbf{V}, \gamma)$ be a CFA-view of $\mathbf{D}$. Then $(M_1, M_2) \in \mathsf{Congr}(\overline{\Gamma})$ iff the following two dual conditions are satisfied.*

$(\forall A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_2))(A_1, A_2) \in \mathsf{Congr}(\Gamma))$

$(\forall A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_1))(A_1, A_2) \in \mathsf{Congr}(\Gamma))$

oarieli@mta.ac.il

(b) *Let* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ *be CFA-views of* $\mathbf{D}$. *Then* $(M_1, M_2) \in \mathsf{Congr}(\overline{\Gamma}_1) \circ \mathsf{Congr}(\overline{\Gamma}_2)$ *iff the following two dual conditions are satisfied.*

$$(\forall A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_2))(A_1, A_2) \in \mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2))$$

$$(\forall A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_1))(A_1, A_2) \in \mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2))$$

Proof. Part (a) is a direct consequence of conditions (atg:1) and (atg:2) of Definition 4.3 and the definition of completion. To establish (b), let $M_1, M_2 \in \mathsf{DB}(\mathbf{D})$ satisfy the two dual conditions of (b). For each $A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1)$, let $A_2 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_2)$ with $(A_1, A_2) \in \mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2)$, and let $N_{A_1} \in \mathsf{LDB}(\mathbf{D})$ be such that $(A_1, N_{A_1}) \in \mathsf{Congr}(\Gamma_1)$ and $(N_{A_1}, A_2) \in \mathsf{Congr}(\Gamma_2)$. N_{A_1} may furthermore be chosen to be a member of $\mathsf{ExtAtoms}(\mathbf{D})$, since every $A \in \mathsf{ExtBasis}_{\mathbf{D}}(N_{A_1})$ must be equivalent to A_1 under $\mathsf{Congr}(\overline{\Gamma}_1)$ and equivalent to A_2 under $\mathsf{Congr}(\overline{\Gamma}_2)$, in view of (a). Thus, if $N_{A_1} \neq \bot_{\mathbf{D}}$, any element of $\mathsf{Basis}_{\overline{\mathbf{D}}}(N_{A_1})$ will serve as well as N_{A_1} itself. (If $N_{A_1} = \bot_{\mathbf{D}}$, leave it as is.) Now, again by the characterization of (a), $(A_1, N_{A_1}) \in \mathsf{Congr}(\overline{\Gamma}_1)$ and $(N_{A_1}, A_2) \in \mathsf{Congr}(\overline{\Gamma}_2)$. Dualize this process for each $A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2)$; choose $A_1 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_1)$ with $(A_1, A_2) \in \mathsf{Congr}(\Gamma_1) \circ \mathsf{Congr}(\Gamma_2)$, and let $N_{A_2} \in \mathsf{ExtAtoms}(\mathbf{D})$ be such that $(A_1, N_{A_2}) \in \mathsf{Congr}(\Gamma_1)$ and $(N_{A_2}, A_2) \in \mathsf{Congr}(\Gamma_2)$. Put $N' = \bigcup(\{N_{A_1} \mid (A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1)\} \cup \{N_{A_2} \mid (A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2)\})$. Then $(M_1, N') \in \mathsf{Congr}(\overline{\Gamma}_1)$ and $(N', M_2) \in \mathsf{Congr}(\overline{\Gamma}_2)$, whence $(M_1, M_2) \in \mathsf{Congr}(\overline{\Gamma}_1) \circ \mathsf{Congr}(\overline{\Gamma}_2)$. The converse condition follows immediately from part (a). □

Proposition 4.12 (Extension of commuting congruences). *Let* $\mathbf{D}$ *be a CFA-schema, and let* $\{\Gamma_1, \Gamma_2\}$ *be a fully commuting pair of CFA-views of* $\mathbf{D}$. *Then* $\Gamma_1 \wedge \Gamma_2$ *is a CFA-view of* $\mathbf{D}$, *and* $\{\overline{\Gamma}_1, \overline{\Gamma}_2\}$ *is a fully commuting pair of views of* $\overline{\mathbf{D}}$, *with* $\overline{\Gamma}_1 \wedge \overline{\Gamma}_2 = \overline{\Gamma_1 \wedge \Gamma_2}$. □

It is now possible to extend the notions of absolute k-models of Definition 3.6 to relative notions, and to prove the main theorem. A relative (to meet complement $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$) k-model in the view $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ is a k-model of $\mathbf{V}_1$ whose $\Gamma_1 \wedge \Gamma_2$ component is already a legal database of $\mathbf{V}_{1\ \gamma_1 \wedge_{\gamma_2}} \mathbf{V}_2$. Such models are central to the update process because the property of the $\Gamma_1 \wedge \Gamma_2$ component being legal does not change under constant-complement update, since that component of the state is held constant. The main theorem then asserts that, for the view to have this property, it suffices that the main schema have the k-submodel property.

Definition 4.13 (Relative k-models). *Let* $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ *be a CFA-schema, let* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ *be CFA-views of* $\mathbf{D}$ *with* $\Gamma_2 \leq \Gamma_1$ *(i.e., with a morphism* $f : \Gamma_1 \to \Gamma_2$*) , and let* $k \in \mathbb{N}$.

(a) *The database* $M \in \mathsf{DB}(\mathbf{V}_1)$ *is called* Γ_2-legal *if* $\lambda\langle\overline{\Gamma}_1, \overline{\Gamma}_2\rangle(M) \in \mathsf{LDB}(\mathbf{V}_2)$, *and it is called a* Γ_2-relative k-model *for* $\mathbf{V}_2$ *if it is both* Γ_2*-legal and a* k*-model of* $\mathbf{V}_1$.

(b) *The view* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *has the* Γ_2-relative k-submodel property *if, for every* $M \in \mathsf{DB}(\mathbf{V}_1)$, $M \in \mathsf{LDB}(\mathbf{V}_1)$ *iff it is a* Γ_2*-relative* k*-model for* $\mathbf{V}_1$.

oarieli@mta.ac.il

Theorem 4.14 (Preservation of complexity). *Let* $\mathbf{D}$ *be a CFA-database schema, and let* $\{\Gamma_1 = (\mathbf{V}_1, \gamma_1), \Gamma_2 = (\mathbf{V}_2, \gamma_2)\}$ *be a subdirect complementary pair of CFA-views of* $\mathbf{D}$*. Let* $k \in \mathbb{N}$*. If* $\mathbf{D}$ *has the* k*-submodel property, then* Γ_1 *has the* $(\Gamma_1 \wedge \Gamma_2)$*-relative* k*-submodel property.*

Proof. First of all, note that γ_1, $\overline{\gamma}_1$, γ_2, and $\overline{\gamma}_2$ are downwardly strong and injectively generating, and that $\mathbf{V}_1$, $\overline{\mathbf{V}}_1$, $\mathbf{V}_2$, and $\overline{\mathbf{V}}_2$ are closed under subinstances, in view of Observation 3.7 and Proposition 3.11.

Let $M_1 \in \mathsf{DB}(\mathbf{V}_1)$ be a $(\Gamma_1 \wedge \Gamma_2)$-relative k-model; the goal is to establish that $M_1 \in \mathsf{LDB}(\mathbf{V}_1)$. To this end, begin by choosing $M_2 \in \mathsf{LDB}(\mathbf{V}_2)$ with the property that $\lambda\langle\overline{\Gamma}_1, \overline{\Gamma}_1 \wedge \overline{\Gamma}_2\rangle(M_1) = \lambda\langle\overline{\Gamma}_2, \overline{\Gamma}_1 \wedge \overline{\Gamma}_2\rangle(M_2) = \lambda\langle\Gamma_2, \Gamma_1 \wedge \Gamma_2\rangle(M_2)$. Such a choice of M_2 is possible since the view morphism $\lambda\langle\overline{\Gamma}_2, \overline{\Gamma}_1 \wedge \overline{\Gamma}_2\rangle$ is surjective. Observe that $(M_1, M_2) \in \overline{\mathbf{V}}_1 \,{}_{\overline{\gamma}_1}\!\otimes_{\overline{\gamma}_2} \overline{\mathbf{V}}_2$. Define $M = (\overline{\gamma}_1 \otimes \overline{\gamma}_2)^{-1}(M_1, M_2)$. Choose $N \in \mathsf{DB}(\mathbf{D})$ such that $N \subseteq M$ with two properties; first, that $\overline{\gamma}_1(N) = \overline{\gamma}_1(M)$, and, second, that $\overline{\gamma}_1$ is injective on $\mathsf{Basis}_{\overline{\mathbf{D}}}(N)$; i.e., $A_1, A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(N)$ and $\overline{\gamma}_1(A_1) = \overline{\gamma}_1(A_2)$ implies that $A_1 = A_2$. Such a choice is possible since $\overline{\gamma}_1$ is injectively generating. Now $(\overline{\gamma}_1 \otimes \overline{\gamma}_2)(N) = (M_1, M_2')$, with $M_2' \in \mathsf{LDB}(\mathbf{V}_2)$. ($M_2' \in \mathsf{LDB}(\mathbf{V}_2)$ since $M_2 \in \mathsf{LDB}(\mathbf{V}_2)$ and $\mathbf{V}_2$ is closed under subinstances.) Next, let $M'' \in \mathsf{DB}(\mathbf{D})$ with $M'' \subseteq N$ and $\mathsf{Card}(M'') \leq k$. Define $(M_1'', M_2'') = (\overline{\gamma}_1 \otimes \overline{\gamma}_2)(M'')$. $M_2'' \in \mathsf{LDB}(\mathbf{V}_2)$ since $M_2'' \subseteq M_2'$ and $\mathbf{V}_2$ is closed under subinstances, and $M_1'' \in \mathsf{LDB}(\mathbf{V}_1)$ since $M_1'' \subseteq M_1$, $\mathsf{Card}(M_1'') = \mathsf{Card}(M'') \leq k$, and M_1 is a relative $(\Gamma_1 \wedge \Gamma_2)$-model. Thus, $M'' \in \mathsf{LDB}(\mathbf{D})$, since $\overline{\gamma}_1 \otimes \overline{\gamma}_2$ is an isomorphism. However, M'' was an arbitrary submodel of N in $\mathsf{DB}(\mathbf{D})$ of size at most k; thus, since $\mathbf{D}$ has the k-submodel property, $N \in \mathsf{LDB}(\mathbf{D})$. Finally, this implies that $M_1 = \gamma_1(N) \in \mathsf{LDB}(\mathbf{V}_1)$; whence Γ_1 has the $(\Gamma_1 \wedge \Gamma_2)$-relative k-model property. □

Example 4.15. It is instructive to give a detailed example at this point which illustrates the ideas of relative complexity. The example is $\mathbf{E}_2$, which was already introduced in Section 1. Specifically, let $\mathbf{E}_2$ denote the relational schema on five attributes with the single relation symbol $S[ABCDE]$. It is constrained by the set $\mathcal{F}_2 = \{A \to D, B \to D, CD \to A, A \to E\}$ of FDs. Since this schema is constrained by FDs, it clearly has the 2-submodel property. The view to be updated is $\Pi_{ABCE} = (S[ABCE], \pi_{ABCE})$, while the complement to be held constant is $\Pi_{ABCD} = (S[ABCD], \pi_{ABCD})$. The pair $\{\Pi_{ABCE}, \Pi_{ABCD}\}$ is lossless, since the dependency $A \to E$ implies the join dependency $ABCD \bowtie ABCE$, and it is dependency preserving since every FD in $\mathcal{F}_2$ embeds into one of the two views. Thus, it forms a meet-complementary pair, with meet $\Pi_{ABC} = (S[ABC], \pi_{ABC})$ [2, 2.16], [3, 2.17]. The updates which are allowed on Π_{ABCE} are precisely those which hold Π_{ABC} constant; that is, those which change only the E-value of a tuple. In view of the above theorem, Π_{ABCE} has the Π_{ABC}-relative 2-submodel property, since the main schema $\mathbf{E}_2$ has the 2-submodel property. Note that this is the case even though the view Π_{ABCE} cannot be finitely axiomatizable [4].

oarieli@mta.ac.il

5 Update Strategies

To complete the transition to the CFA-context, the connection between the results of the previous section and formal update strategies must be made. For the most part, the approach is similar to that taken in [2] and [3]; however, an adjustment is necessary to ensure that the complement view generated by an update strategy is a CFA-view.

Summary 5.1 (Augmenting update strategies for CFA-views). To adapt the conditions (upt:1)-(upt:8) summarized in Summary 2.1 and Summary 2.2 to the CFA-context, it is necessary to ensure that the equivalence $\equiv_\rho$ of an update strategy ρ is in fact an atomically generated equivalence, so that the ρ-complement $\widetilde{\Gamma}^\rho$ of the CFA-view Γ is in fact a CFA-view. The appropriate addition to (upt:1)-(upt:8) is the following.

(upt:9) If $\rho(M_1, \gamma(M_2)) = M_2$, then
$(\forall A_1 \in \mathsf{Basis}_{\mathbf{D}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\mathbf{D}}(M_2))(\rho(A_1, \gamma(A_2)) = A_2)$

An update strategy ρ which satisfies all of (upt:1)-(upt:9) will be called a *CFA-update strategy*. Essentially, this means that every update is composed of updates on the underlying family of atoms. It is easy to see that (upt:9) holds in the classical setting of the lossless and dependency-preserving decomposition of a relational schema, as elaborated in [2, 2.15 and 2.16], [3, 2.16 and 2.17].

Lemma 5.2. *The induced update family $\equiv_\rho$ is an atomically generated equivalence iff ρ is a CFA-update strategy.*

Proof. Follows from Definition 4.3 and Proposition 4.5. □

Now the "CFA" equivalent of [2, 3.9] and [3, 3.10] follows directly.

Theorem 5.3. *Let $\mathbf{D}$ be a CFA-schema, and let Γ be a CFA-view of $\mathbf{D}$. There is natural bijective correspondence between CFA-update strategies for Γ and meet complements of that view which are also CFA-views. Specifically:*

(a) *For any CFA-update strategy ρ, $\mathsf{UpdStr}\langle \Gamma, \widetilde{\Gamma}^\rho \rangle = \rho$.*

(b) *For any meet complement Γ_1 of Γ which is also a CFA-view, $\widetilde{\Gamma}^{\mathsf{UpdStr}\langle \Gamma, \Gamma_1 \rangle} = \Gamma_1$.* □

Notation 5.4 (Notational convention). Throughout the rest of this section, unless stated specifically to the contrary, let $\mathbf{D} = (\mathsf{LDB}(\mathbf{D}), \subseteq)$ be a CFA-schema, $\Gamma = (\mathbf{V}, \gamma)$ a CFA-view of $\mathbf{D}$, U and T closed update families for $\mathbf{D}$ and $\mathbf{V}$, respectively, and ρ a CFA-update strategy for T with respect to U.

Definition 5.5 (The completion of an update strategy).

(a) The *completion* of U, denoted $\overline{U}$, is the relation on $\mathsf{DB}(\mathbf{D}) \times \mathsf{DB}(\mathbf{D})$ defined by $(M_1, M_2) \in \overline{U}$ iff the following two (dual) conditions are satisfied.

(i) $(\forall A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_2))(A_1, A_2) \in U)$

(ii) $(\forall A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_1))(A_1, A_2) \in U)$

oarieli@mta.ac.il

(b) The *completion* of ρ is the function $\overline{\rho} : \mathsf{DB}(\mathbf{D}) \times \mathsf{DB}(\mathbf{V}) \to \mathsf{DB}(\mathbf{D})$ given by $(M_1, N_2) \mapsto M_2$ iff $\overline{\gamma}(M_2) = N_2$ and the following two (dual) conditions are satisfied:

(i) $(\forall A_1 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_1))(\exists A_2 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_2))(\rho(A_1, \gamma(A_2)) = A_2)$

(ii) $(\forall A_2 \in \mathsf{Basis}_{\overline{\mathbf{D}}}(M_2))(\exists A_1 \in \mathsf{ExtBasis}_{\overline{\mathbf{D}}}(M_1))(\rho(A_1, \gamma(A_2)) = A_2)$

Lemma 5.6. *$\overline{\rho}$ is a CFA-update strategy for $\overline{T}$ with respect to $\overline{U}$.*

Proof. This is a routine verification against the conditions (upt:1)-(upt:9). The details are omitted. □

In [2, 4.2], [3, 4.3], it is established that there is only one way to reflect an update on a closed view back to the main schema, provided that update is realizable as sequence of legal insertions and deletions. Using the framework developed in this paper, it is possible to drop the condition of legality on the intermediate states; in other words, the reflection of the view update back to main schema is unique as long as it is realizable as sequence of insertions and deletions, even though the intermediate states may not be legal. In other words, for all practical purposes, there is only one way to reflect an update under a closed update strategy back to the main schema, regardless of whether or that update is order realizable. The formal details follow.

Definition 5.7 (Syntactic order-based updates). Following [2, 4.1] and [3, 4.1], a pair $(M_1, M_2) \in U$ is called a *formal insertion* with respect to U if $M_1 \leq_{\mathbf{D}} M_2$; a *formal deletion* with respect to U if $M_2 \leq_{\mathbf{D}} M_1$; and an *order-based update* with respect to U if there exists a nonempty sequence $(N_1, N_2), (N_2, N_3), \ldots, (N_{k-2}, N_{k-1}), (N_{k-1}, N_k)$ of elements of U with the properties that $N_1 = M_1$, $N_k = M_2$, and each pair (N_i, N_{i+1}), $1 \leq i \leq k-1$, is either a formal insertion or else a formal deletion with respect to U. The update family U is called *order realizable* if every pair in U is an order-based update.

More generally, call a pair $(M_1, M_2) \in U$ a *syntactic order-based update* if (M_1, M_2) is an order-based update in $\overline{U}$, and call U *syntactically order realizable* if every pair in U is a syntactic order-based update. Since $\overline{\rho}$ is an update strategy by Lemma 5.6, the following extension of [2, 4.2] and [3, 4.3] follows immediately.

Theorem 5.8 (uniqueness of reflection of syntactic order-based view updates). *Let ρ_1 and ρ_2 be update strategies for T with respect to U. Then, for any $M \in \mathsf{LDB}(\mathbf{D})$ and $N \in \mathsf{LDB}(\mathbf{V})$ with $(\gamma(M), N) \in T$ a syntactic order-based update, it must be the case that $\rho_1(M, N) = \rho_2(M, N)$. In particular, if T is syntactically order realizable, then $\rho_1 = \rho_2$.* □

Example 5.9. Let $\mathbf{E}_3$ be the relational schema with a single relation symbol $R[ABC]$ on three attributes, constrained by the set $\mathcal{F}_3 = \{B \to A, B \to C\}$. Let the view to be updated be $\Pi_{AB} = (R[AB], \pi_{AB})$, and the complement to be held constant $\Pi_{BC} = (R[BC], \pi_{BC})$. In view of [2, 2.16], [3, 2.17], these two views form a meet complementary pair with meet $\Pi_B = (R[B], \pi_B)$. The updates which are allowed on Π_{AB} are those which hold the projection on B

oarieli@mta.ac.il

constant; since the FD $B \to A$ holds as well, this means that the only updates which are possible are those which change the A-value of existing tuples. These are not order-based updates; therefore, the main theorem [2, 4.2] [3, 4.3] does not provide a direct guarantee of the uniqueness of their translations. However, when the integrity constraint $B \to A$ is ignored, the resulting update family is syntactically order based, and so Theorem 5.8 guarantees a unique translation of all such updates on Π_{AB}, regardless of whether or not the complement to be held constant is Π_{BC}. Indeed, since the updates to Π_{AB} which hold Π_B constant are syntactically order realizable, the update strategy obtained by holding Π_{BC} constant is the only one possible.

This elegant solution should be contrasted with the rather complex and ad hoc approach to establishing uniqueness for the same example in [2, 4.5], [3, 4.6].

6 Final Remarks

Discussion 6.1 (Conclusions and proposed future work). It has been shown that, under quite general conditions, the explosion in constraint complexity which may occur when moving from a main schema to a view cannot adversely affect the complexity of updates issued against a closed database view. Essentially, such explosions in complexity must be encapsulated within the meet of the view to be updated and the complement used to define the update strategy. Since that part of the view is not alterable during an update, the complexity of the constraints on the meet is irrelevant. The complexity which is passed along to the view-update process is no greater than the corresponding complexity on the main schema.

The scope of the approach presented here is limited to a context which generalizes EGDs of the relational model, and covers neither TGDs such as join dependencies nor non-universal dependencies such as foreign-key constraints. In terms of practical use, the most salient task is to extend the framework to include foreign-key dependencies, since they are used in real, commercial relational database systems. To accomplish this, it seems necessary to extend the notion of a CFA-schema to one which explicitly recaptures the idea of a multi-relation schema, since such dependencies involve multiple relations in a fundamental way.

Extension to recapture TGDs is more straightforward, involving a generalization of the notion of k-model to (k_1, k_2)-model, with $k_1 \in \mathbb{N}$ and $k_2 \geq 1$ a real number. Roughly, $M \in \mathsf{DB}(\mathbf{D})$ is a (k_1, k_2)-model if there is a k_1-model $M' \in \mathsf{DB}(\mathbf{D})$ with $M \subseteq M'$ and $\mathsf{Card}(M') \leq k_2 \cdot \mathsf{Card}(M)$. Note that k-models are just $(k, 1)$-models in this extended context. Extension to recapture views defined by joins is also reasonably straightforward. While the view mappings are obviously no longer basis preserving, it is nonetheless possible to establish the necessary properties (i.e., those of Definition 3.10 and Proposition 3.11). All of these topics will be addressed in a forthcoming full version of this paper.

Finally, since the theory is not tied to any particular data model, it seems appropriate to apply this theory to models other than the classical relational. The difficulty is to find a suitable starting point, since the type of complexity

oarieli@mta.ac.il

questions addressed here have not been studied in any detail for models other than the relational.

Discussion 6.2 (Relationship to other work). In an early paper, Cosmadakis and Papadimitriou [14] present pessimistic complexity results which would appear to contradict those obtained here. However, they work with general subdirect complements, and not meet complements, and so their results do not apply to the closed update strategies considered here. They also investigate the complexity of identifying a minimal (not necessarily meet) complement which will support a given update, again with pessimistic results. Recently, Lechtenbörger and Vossen [15] have also looked at the complexity of the problem of identifying (not necessarily meet) complements to views, but for the purpose of identifying information missing in the view, and not with an eye towards update strategies. Their approach, by design, does not concern itself with meet complements or update strategies. Beyond those works, most of the literature on the problem of complexity of view updates is focused on logic databases. The fundamental issues which arise in that context (theory-oriented database models) are quite different from those of instance-oriented database models, and so a meaningful comparison is difficult at best.

References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Systems **6** (1981) 557–575
2. Hegner, S.J.: Uniqueness of update strategies for database views. In: Foundations of Information and Knowledge Systems: Second International Symposium, FoIKS 2002, Salzau Castle, Germany, February 2002, Proceedings, Springer-Verlag (2002) 230–249
3. Hegner, S.J.: An order-based theory of updates for database views. Ann. Math. Art. Intell. **40** (2004) 63–125
4. Hegner, S.J.: A simple counterexample to the finite axiomatizability of relational views. unpublished note, available at the author's web site (2003)
5. Adámek, J., Herrlich, H., Strecker, G.: Abstract and Concrete Categories. Wiley-Interscience (1990)
6. Langerak, R.: View updates in relational databases with an independent scheme. ACM Trans. Database Systems **15** (1990) 40–66
7. Hegner, S.J.: Foundations of canonical update support for closed database views. In Abiteboul, S., Kanellakis, P.C., eds.: ICDT'90, Third International Conference on Database Theory, Paris, France, December 1990, Springer-Verlag (1990) 422–436
8. Hegner, S.J.: Characterization of desirable properties of general database decompositions. Ann. Math. Art. Intell. **7** (1993) 129–195
9. Hegner, S.J.: Unique complements and decompositions of database schemata. J. Comput. System Sci. **48** (1994) 9–57
10. Davey, B.A., Priestly, H.A.: Introduction to Lattices and Order. second edn. Cambridge University Press (2002)
11. Grätzer, G.: General Lattice Theory. Second edn. Birkhäuser Verlag (1998)

oarieli@mta.ac.il

12. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
13. Wang, K., Graham, M.H.: Constant-time maintainability: A generalization of independence. ACM Trans. Database Systems **17** (1992) 201–246
14. Cosmadakis, S., Papadimitriou, C.: Updates of relational views. J. Assoc. Comp. Mach. **31** (1984) 742–760
15. Lechtenbörger, J., Vossen, G.: On the computation of relational view complements. ACM Trans. Database Systems **28** (2003) 175–208

oarieli@mta.ac.il

Equivalence of OLAP Dimension Schemas

Carlos A. Hurtado and Claudio Gutiérrez

Department of Computer Science
Universidad de Chile
{churtado,cgutierr}@dcc.uchile.cl

Abstract. Dimension schemas are abstract models of the data hierarchies that populate OLAP warehouses. Although there is abundant work on schema equivalence in a variety of data models, these works do not cover dimension schemas. In this paper we propose a notion of equivalence that allows to compare dimension schemas with respect to their information capacity. The proposed notion is intended to capture dimension schema equivalence in the context of OLAP schema restructuring. We offer characterizations of schema equivalence in terms of graph and schema isomorphisms, and present algorithms for testing it in well known classes of OLAP dimension schemas. Our results also permit to compare the expressiveness of different known classes of dimension schemas.

1 Introduction

OLAP dimensions are data hierarchies that populate data warehouses. These entities are hierarchically organized information that define the perspective upon which the data is viewed. As an example, in a data warehouse we may have dimensions describing products, stores and time, which may be used to visualize the facts generated by a sales process.

Figure 1 depicts a dimension that models financial services offered by a bank: accounts, credit cards and loans. On the left hand side of Figure 1, there is a graph called *hierarchy schema* which models the structure of the dimension. The vertices of this graph are called categories. On the right hand side, there is another graph, called *hierarchy domain*, whose vertices, called members, are grouped by categories and ordered by a child/parent relation. For example, in the dimension at hand, we may say that member $p1$ belongs to the category *Product* and $p1$ has $d1$ as a parent in the category *Department*.

In the dimension at hand, some types of products, like personal loans and some sorts of accounts, are handled by branches, whereas others, like mortgage and corporate loans, are handled by departments. Only the products in branches are classified through the hierarchy path *Product-ProdType-ProdClass-All*. There is a manager in charge of each branch and department. Finally, it happens that the *Asia* branch and all departments handle products in only one category; thus, their managers belongs to a member in *ProdClass*.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 176–195, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

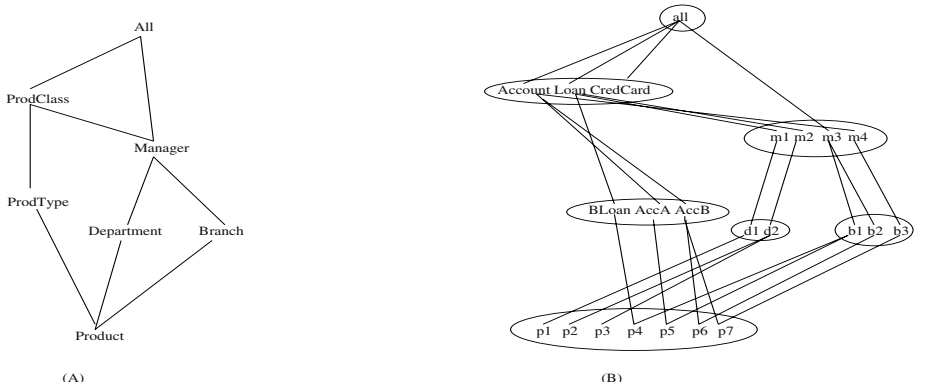


Fig. 1. The dimension Product: (A) hierarchy schema; (B) child/parent relation.

1.1 Dimension Schemas

A *dimension schema* is an abstract model of a dimension commonly used to support summarizability reasoning in OLAP applications [HM02], that is, to test whether aggregate views defined for some categories can be correctly derived from a set of precomputed views defined for other categories. A dimension schema, being an abstract representation of a dimension, represents the set of possible dimensions that conforms to it. This set reflects the *information capacity* of the schema. Thus when we perform reasoning on the schema, we infer properties of all the dimensions in the set.

A central drawback of traditional dimension schemas is that they do not account for structural heterogeneity. Such schemas model dimensions in which members in a category c should have a parent in every category directly above c, a condition we refer to as *homogeneity*. This restriction is unnatural since in many application domains the members of a category have parents (resp., ancestors) in different sets of categories. As an example, in the hierarchy domain of Figure 1 (B), some products are under branches while some others are under departments.

In previous work [HM02] we introduced semantically rich dimension schemas to support summarizability reasoning in heterogeneous dimensions. In our setting, dimension schemas are modeled as a hierarchy schema along with a set of integrity constraints, called *dimension constraints*. The hierarchy schema represents a set of links for the child/parent relation, that is, whenever we have a child/parent relationship between two members in two categories, the categories must be directly connected in the hierarchy schema. Dimension constraints are statements that specify legal paths allowed in the hierarchy domain. The constraints are used to place further restrictions to let the schema capture more precisely different sets of dimensions.

For example, we may require that all the products handled by some branch are not handled by departments, and vice versa. This is stated by

oarieli@mta.ac.il

the constraint saying that each product can have ancestors in either the path $\langle Product, Branch\rangle$ or the path $\langle Product, Department\rangle$, but not in both. Other constraints may express that the ancestor of some members rollup to members that form a particular path in the hierarchy schema. For example we model that "the manager of the Asia branch rolls up to a product class (because the manager handles products that belong to a single product class)" as "$\langle Branch = Asia\rangle \Leftrightarrow \langle Branch, Manager, ProdClass\rangle$". The expressions in brackets are atomic statements (called *atoms*). It turns out that Boolean combinations of atoms are needed to support summarizability reasoning [HM02].

Simple forms of these constraints characterize typical classes of OLAP schemas. For example, the condition of homogeneity of the edge $(ProdType, ProdClass)$ can be expressed with the constraint $\langle ProdType, ProdClass\rangle$, which asserts that each product type belongs to a product class. In this sense, the class of schemas with dimension constraints subsumes other classes of schemas in OLAP (see Section 2.4) such as the dimension schemas of Jagadish el al. [JLS99], called in this paper *canonical*, which partially solves the limitations of traditional OLAP models by allowing several bottom categories, but keeping the homogeneity restriction. Canonical schemas allow unbalancedness, that is, they can have several bottom categories. In this form, members in different bottom categories may have ancestors in different hierarchy paths in the schema.

Different classes of dimension schemas are classified in [Hur02] where for "traditional OLAP" we refer to the basic class of homogeneous schemas with a single bottom category (balanced schemas).

1.2 Problem Statement

Similarly to the case of general database schemas, two dimension schemas can be compared with respect to their information capacity. Schemas with the same information capacity can be used to simulate each other. In a typical modeling scenario the user starts with some schema and proceeds to restructure it. In the context of OLAP, it is very important that the restructuring process preserves schema equivalence because the schema is more useful for reasoning about data than it is just as a container of data. So we would like to keep the information on the schema as precise as possible to capture the set of instances as tight as possible.

The goal of this paper is to study dimension schema equivalence in the context of OLAP schema restructuring. Formal notions of schema equivalence are fundamental to sit restructuring techniques on solid grounds. For example, Miller et al. [MIR94] argue that the restructuring task may be addressed following two different strategies: (i) build a desired schema and then test whether it is equivalent to the original schema; (ii) use a set of primitives to transform the original schema into a desired schema. In both approaches, we need to define under which conditions two dimension schemas are equivalent. In the first approach we need algorithms for the equivalence test. The second approach requires a set of well defined dimension transformations. The central desirable properties of

oarieli@mta.ac.il

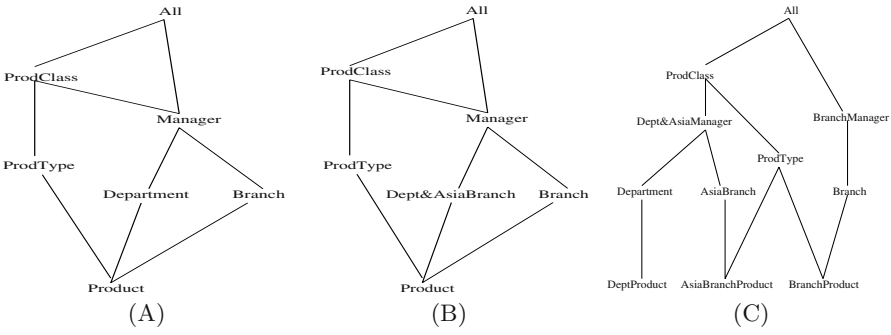


Fig. 2. Product hierarchy schemas.

such a set, soundness and completeness [Alb00], depend on the notion of schema equivalence used as well.

We cast the restructuring task as a process in which the structure of the dimension (i.e. its hierarchy schema) changes but its data hierarchy (hierarchical domain) does not.

Example 1. Consider the hierarchy schemas shown in Figure 2. The three hierarchy schemas can be used to model the hierarchy domain of Figure 1. Hierarchy schema (A) is the same as the one of Figure 1. Hierarchy schema (B) has Asia branch grouped with departments in a single category. Finally, in hierarchy schema (C), the products (bottom members) are split into three categories: *DeptProduct*, *AsiaBranchProduct*, and *BranchProduct*, the branches are split into *Branch* and *AsiaBranch*, and the managers are split into the categories *Dep&AsiaManager* and *BranchManager*. Notice that the dimension having hierarchy schema (C) along with the hierarchy domain of Figure 1 is homogeneous.

Schema equivalence has been formalized by requiring the existence of a bijective mapping between the instances of two equivalent schema [Hul86]. Example 1 shows that a great deal of flexibility in OLAP dimension modeling can be captured by restructuring processes in which members are reorganized into different categories but their hierarchy domain does not change. Thus at the schema level, the correctness of a restructuring process may be formalized by requiring the existence of bijective instance mappings between the schemas which preserve hierarchy domains. As members are associated with facts in datacubes, this mapping restriction guarantees that the aggregate data are preserved through different dimension instances in the restructuring process, thus avoiding aggregate data re-computations, and keeping users to browse aggregate data using the same hierarchy domain.

Example 2. Consider the following dimension schemas. The dimension schema `productA` has the hierarchy schema of Figure 1 (A) along with the dimension constraints (a)-(d) of Figure 3. The dimension schema `productB` has the hierarchy schema of Figure 1 (B) along with the dimension constraints (a')-(c') of

oarieli@mta.ac.il

(a) $\langle Product, Branch\rangle \oplus \langle Product, Department\rangle$
(b) $\langle Product, Branch\rangle \Leftrightarrow \langle Product, ProdType\rangle$
(c) $\langle Department, Manager, ProdClass\rangle$
(d) $\langle Branch = Asia\rangle \Leftrightarrow \langle Branch, Manager, ProdClass\rangle$
(a') $\langle Product, Dept\&AsiaBranch\rangle \oplus \langle Product, Branch\rangle$
(b') $(\langle Product, Branch\rangle \vee \langle Product, .., Dept\&AsiaBranch = Asia\rangle \Leftrightarrow \langle Product, ProdType\rangle$
(c') $\langle Dept\&AsiaBranch, Manager, ProdClass\rangle$
(e) $\langle c, c'\rangle$ for each edge (c, c') in the hierarchy schema
(f) $\langle AsiaBranch = Asia\rangle$

Fig. 3. Dimension Constraints for the product hierarchy schemas.

Figure 3. Finally, the dimension schema `productC` has the hierarchy schema of Figure 1 (C) along with the constraint (f) of Figure 3, and a constraint $\langle c, c'\rangle$, for every edge (c, c') in the hierarchy schema. Observe that the products in schema `productC` are now split in three categories depending on where they roll up to (only to Department, to AsiaBranch and ProdType, etc.)

The constraints make the different schemas equivalent (although we have not proved this yet). For example, constraints (c) and (d) of `productA` translate to (c') in `productB`.

1.3 Related Work

There has been abundant work on OLAP dimension modeling over the past few years [CT97,HMV99,LAW98,PJE99,JLS99]. However, to the best of our knowledge, there are no studies regarding dimension schema equivalence. Several notions of schema equivalence for a variety of data models have been proposed. The most general notion of schema equivalence, *absolute equivalence* [Hul86] characterizes the minimum requirements that two schemas must satisfy in order for them to have the same information capacity. Absolute equivalence is formalized by requiring the existence of a bijection between the instances of the schemas. Any arbitrary mapping may be used to guarantee absolute equivalence. In addition, the mappings are not required to be finitely specifiable (they can be an infinite list of pairs of schema instances). A hierarchy of more restricted notions of equivalence has been proposed [Hul86]. For example: *internal equivalence* requires the existence of a bijection that neither creates nor destroy elements in the instances; *query equivalence* requires the mappings to be expressible in the query language of the data model. Other notions of equivalence and their testing have been studied for generic graph data models by Miller et al. [MIR94] and nested data models [VL00]. Our notion of equivalence places minimum restrictions to the instance mappings in order to let them capture all possible ways of grouping members into categories in a schema.

A detailed description of the relationship of dimension constraints and the other constraints for OLAP and other data models is presented by Hurtado

oarieli@mta.ac.il

[HGM03]. Next, we highlight the most important points in this respect. As explained in several papers (e.g., [JLS99,HMV99]) OLAP dimension may be modeled as a set of normalized tables, one for each category, containing the rollup mappings that start from the category, along with the the attributes of the category. Therefore dimension schemas may be formalized using a relational database setting. It is easily verified that dimension constraints are FOL constraints; therefore, our entire framework is a fragment of FOL. Abiteboul et al. [AV97] study a class of FOL constraints called *embedded constraints* that formalize a wide variety of constraints studied in the database literature. They essentially express that the presence of some tuples in the instance implies the presence of some other tuples in the instance or implies that certain tuple components are equal. Dimension constraints cannot be expressed with embedded constraints, since we cannot assert with them that "some tuples or some other tuples appear in the instance", which are needed to characterize summarizability. Dimension Constraints restrict data in a similar fashion to disjunctive existential constraints (dec's) [Gol81] (which are not embedded constraints). Disjunctive existential constraints are used to characterize the possible sets of non-null attributes that may occur in the tuples of a relation; conceptually, the possible objects that are mixed in the relation. Another class of constraints along the same lines is presented by Husemann et al. [HLV00]. These constraints can be easily represented with dimension schemas, and do not have the full expressiveness of the Boolean connectives needed for summarizability reasoning. *Path constraints* [AV97,BFS98] allow describing certain forms of heterogeneity in semistructured data. They characterize the existence of paths associated with sequences of labels in semistructured data. However, path constraints also lack the entire expressiveness needed to characterize summarizability, and to describe the sort of heterogeneity arising in OLAP applications. On the other hand, path constraints are interpreted over data which have fewer restrictions in their structure than OLAP dimensions, yielding to a different treatment and complexity of their inference.

1.4 Contributions

This paper presents the following contributions:

- A notion of equivalence, hierarchical equivalence, which allows comparison of dimension schemas with respect to their information capacity.
- A proof that hierarchical equivalence can be characterized in terms of graph and schema isomorphisms in two known classes of dimension schemas, called here canonical and balanced. The formal proof of this intuitive connection is non-obvious, as we show in Section 4. The result proves that canonical schemas are more expressive than balanced schemas, hence formally justifying the introduction of canonical schemas.
- A class of schemas –frozen schemas– that act as normal forms for dimensions schemas, in the sense that any dimension schema can be reduced via some well defined transformation to a unique (up to isomorphism) frozen

oarieli@mta.ac.il

schema. It is proved that hierarchy equivalence test for frozen dimension reduces to a simple form of schema isomorphism. This result leads to other important property of dimension schemas, namely, that heterogeneous dimension schemas can always be transformed into homogeneous schemas. We sketch an algorithm that performs such transformation in an efficient way, and study its complexity.
– Complexity bounds and a study of algorithmic aspects of hierarchical equivalence testing. In particular, we present a characterization of hierarchical equivalence in terms of mappings between minimal dimensions instance contained by the schemas. This leads to an algorithm for testing hierarchical equivalence. We show that the algorithm is more efficient than testing hierarchical equivalence by reducing the schemas to frozen schemas.

1.5 Outline

The remainder of the paper is organized as follows. In Section 2 we review the main concepts related to schemas and state the notation. Section 3 introduces hierarchical equivalence and show its relation with balanced schemas. Section 4 studies hierarchical equivalence of canonical schemas, and shows that in this context hierarchical equivalence corresponds exactly with graph isomorphism of the corresponding hierarchy schemas. In Section 5 we generalize this result to dimension schemas, that is allowing to compare different hierarchy schemas and constraints. The notion of frozen schema is introduced and studied, along with the algorithmic aspects of hierarchical equivalence are studied. Finally, in Section 6 we briefly conclude and outline further work. The complete proofs are presented in the full version of this paper [HG03].

2 Preliminaries

2.1 Basic Graph Terminology

A (directed) graph G is a pair of sets (V, E) where $E \subseteq V \times V$. Elements $v \in V$ are called *vertices* and pairs $(u, v) \in E$ (directed) *edges*; u and v are *adjacent* vertices. A *path* in G from v to w is a sequence of vertices $v = v_0, \ldots, v_n = w$ such that $(v_i, v_{i+1}) \in E$. We say that v *reaches* w. The *length* of a path is n. A *cycle* is a path with $v = w$. A *dag* is a directed acyclic graph. A *sink* in a dag is a distinguished vertex w reachable from every other vertex in the graph. A *source* in a dag is a distinguished vertex v from which every other vertex of the graph is reachable. A *shortcut* in a dag is a path of length > 1 between two adjacent vertices. Given a vertex v of G, an *upgraph* is the subgraph of G generated by v and all the vertices reachable from it.

Given two graphs $G = (V, E)$ and $G' = (V', E')$, a *graph morphism* is a function $\phi : V \to V'$ preserving edges, that is, $(u, v) \in E$ implies $(\phi(u), \phi(v)) \in E'$. The morphism ϕ is called an *isomorphism (resp. monomorphism, epimorphism)* if ϕ as a function is bijective (resp. injective, onto).

oarieli@mta.ac.il

2.2 Dimension Instance

Assume the existence of (possibly infinite) sets of categories **C**, and of members **M**.

Definition 1 (Hierarchy Schema). *A* hierarchy schema *is a* dag $H = (C, \nearrow)$, *where* $C \subseteq \mathbf{C}$, *having a distinguished category* $\texttt{All} \in C$ *which is a sink.*

Definition 2 (Hierarchy Domain). *A Hierarchy domain is a* dag $h = (M, <)$ *where* $M \subset \mathbf{M}$, *having a distinguished member* $\texttt{all} \in M$ *which is a sink, and without shortcuts.*

The last condition in Definition 2 (no shortcuts) avoids redundancies (transitive edges) in the representation of the data.

Given a child/parent relation $<$, its reflexive and transitive closure, denoted $\leq$, is called *rollup relation*, and is a partial order between members.

Definition 3 (Dimension Instance). *A* dimension instance d over a hierarchy schema $(C, \nearrow)$ *is a graph morphism* $d : (M, <) \to (C, \nearrow)$ *such that:*

1. $(M, <)$ *is a hierarchy domain;*

2. $d(\texttt{all}) = \texttt{All}$*;*

3. $x \leq y \wedge x \leq z$ *implies* $d(y) \neq d(z)$.

The fact that d is a graph morphism in Definition 3 states that whenever we have a child/parent relationship $m_1 < m_2$ between some pair of members $m_1 \in c_1$ and $m_2 \in c_2$, then there is an edge $c_1 \nearrow c_2$ in the hierarchy schema representing links between categories c_1 and c_2. Condition 3 of Definition 3 is a basic restriction in OLAP data modeling [HMV99,CT97,LAW98], and states that the rollup relation $\leq$ is functional (i.e., single valued) between every pair of categories. This motivates to introduce the *rollup mapping* between two categories c_1 and c_2 of a dimension d, denoted $\Gamma_{c_1}^{c_2} d$, which is the restriction of $\leq$ to $d^{-1}(c_1)$ and $d^{-1}(c_2)$.

2.3 Dimension Schema

Next we formalize the notions of dimension constraint and dimension schema.

Definition 4 (Dimension Constraint). *Let* $H = (C, \nearrow)$ *be a hierarchy schema,* $c \in C$, $K \subseteq \mathbf{M}$. *The language of constraints (with root* c*) has the following atoms:*

1. Path atoms: $\langle c, c_1, \cdots, c_n \rangle$, *where* $cc_1 \cdots c_n$ *is a path in* H*;*

2. Equality atoms: $\langle c, .., c' = k \rangle$, *where* c' *is such that there is a path from* c *to* c', *and* $k \in K$.

A dimension constraint with root c *is a Boolean combination* ϕ *of atoms of the above kind.*

Dimension constraints consider the usual connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and $\oplus$ for exclusive disjunction. In addition, $\bot$ and $\top$ will denote the false and the true proposition, respectively.

oarieli@mta.ac.il

Definition 5 (Semantics of Constraints). *Let $d : (M, <) \to (C, \nearrow)$ be a dimension instance, and ϕ a constraint with root c. Then $d \models \phi$ if and only if*

for all $m \in d^{-1}(c)$, $d \models \phi[c/m]$,

where $d \models \phi[c/m]$ is defined recursively as follows:

1. $d \models \langle c, c_1, \ldots, c_n\rangle[c/m]$ iff there is a path $mx_1 \cdots x_n$ in $(M, <)$ with $d(x_i) \in c_i$.

2. $d \models \langle c, .., c' = k\rangle[c/m]$ iff $d(k) \in c'$ and $m \leq k$.

3. $d \models (\phi \wedge \psi)[c/m]$ iff $d \models \phi[c/m]$ and $d \models \psi[c/m]$. Similarly for $\vee$ and the other Boolean connectives.

Given a hierarchy schema H and two sets of constraints Σ, Σ' over H, we say that Σ is equivalent to Σ', if for all dimension instances d over H it holds: $d \models \Sigma$ iff $d \models \Sigma'$.

Now we are ready to introduce the concept of Dimension Schema. The following definition extends Definition 3 in the presence of constraints.

Definition 6 (Dimension Schema). *A* dimension schema *is a pair (H, Σ) where H is a hierarchy schema and Σ is a set of constraints.*

A dimension instance *d over a dimension schema $D = (H, \Sigma)$ is a dimension instance d over H such that $d \models \Sigma$. The set of dimensions instances over D will be denoted by $I(D)$.*

Definition 7 (Schema Equivalence and Isomorphism).

Let $D = (H, \Sigma)$ and $D' = (H', \Sigma')$ be to dimension schemas.

1. D and D' are equivalent*, denoted $D \equiv D'$, iff $H = H'$ and Σ is equivalent to Σ'.*

2. D and D' are isomorphic*, denoted $D \cong D'$, iff there exists a graph isomorphism $f : H \to H'$ such that $(f(H), f(\Sigma)) \equiv (H', \Sigma')$, where $f(\Sigma)$ stands for Σ modulo renaming by f.*

Notice that the notion of equivalence of Definition 7 implies isomorphism.

2.4 Classes of Dimension Schemas

The model we have presented subsumes the dimension models presented in the literature. The following definition formalizes two classes of dimension schemas that arise in OLAP.

Definition 8 (Classes of Dimension Schemas). *Let $D = (H, \Sigma)$ be a dimension schema.*

1. D is canonical *iff H has no shortcuts and Σ is equivalent to $\{\langle c, c'\rangle \mid c \nearrow c'\}$.*

2. D is balanced *iff D is canonical and H has a source.*

A dimension instance d is *homogeneous* if for every pair of categories $c_1 \nearrow c_2$ it holds that the rollup mapping $\Gamma_{c_1}^{c_2} d$ is a total function. Note that the constraint $\langle c, c'\rangle$ where $c \nearrow c'$ forces the rollup mapping from c to c' to be total. Therefore, canonical schemas convey all the homogeneous instances over its hierarchy schema. In this sense, in canonical schemas, Σ captures exactly

oarieli@mta.ac.il

homogeneity. Also notice that we have defined a canonical schema to be shortcut-free, because otherwise Σ would force the categories from which the shortcut start to be empty in every dimension conveyed by the schema. Balanced schemas correspond to the basic class of schemas introduced in early works on OLAP. They are the logical representation of dimension schemas in early snowflake schemas [CD97]. Canonical schemas were introduced by Jagadish et al. [JLS99] to overcome some of the weaknesses of balanced schemas. Canonical schemas allow unbalancedness, that is, they can have dimension instances with two members in the bottom categories having ancestors in different sets of categories. This has been shown to be an important feature to provide flexibility in OLAP modeling.

Example 3. If we delete the constraint (f) to the dimension schema `productC` described in Example 2, the schema turns into a canonical schema.

Given two classes of schemas S_1, S_2, we define $S_1 \sqsubseteq S_2$ iff for each schema in S_1, there is an equivalent schema in S_2. Then it holds Balanced Schemas $\sqsubseteq$ Canonical Schemas $\sqsubseteq$ Dimension Schemas, and the inclusions are proper.

3 Hierarchical Equivalence

In this section we present the notion of hierarchical equivalence in which dimension schemas are related via mappings that preserve the hierarchy domain of the dimensions.

Observe that the notion of schema equivalence of Definition 7 does not allow us to compare schemas having different hierarchy schemas. The following definition generalizes Definition 7 for schemas over arbitrary hierarchy schemas.

Definition 9 (Hierarchical Equivalence). *Two dimension schemas D and D' are* hierarchically equivalent *(h-equivalent) if and only if there is a bijective function $f : I(D) \to I(D')$ such that for all $d \in I(D)$, $dom(d) = dom(f(d))$. In this case we write $D \equiv_h D'$.*

Observe that the relation $\equiv_h$ is an equivalence relation. Also, it is worth noting that the instance mapping f required for h-equivalence is *internal* [Hul86], i.e., it does neither create nor destroy members or constants in the instances. Moreover, the mapping is *generic* [Hul86], that is, given a pair of dimension instances d and d' with $d' = f(d)$, if we apply the same permutation π of members to d and to d', if $\pi(d)$ is in the domain of f then $\pi(d') = f(\pi(d))$. Thus, hierarchical equivalence is a more restricted notion than internal and generic equivalence.

Example 4. Consider the dimension schemas $D_1 = (A, \Sigma_1)$, $D_2 = (B, \Sigma_2)$ and $D_3 = (C, \Sigma_3)$, where A, B, C are the hierarchy schemas in Figure 4, $\Sigma_1 = \Sigma_3 = \emptyset$ and $\Sigma_2 = \{\neg\langle e, f\rangle \vee \neg\langle e, g\rangle\}$. Then $D_1 \equiv_h D_2$ via mapping the members of c to f, the members of d to g, and the members of a and b to e. However, it is not the case that $D_1 \equiv_h D_3$. Indeed, given a member m, there is a unique dimension instance in $I(D_3)$ whose child/parent relation is $\{m < \texttt{all}\}$, but there are two dimension instances in $\mathcal{I}(D_2)$ whose child/parent relation is $\{m < \texttt{all}\}$.

oarieli@mta.ac.il

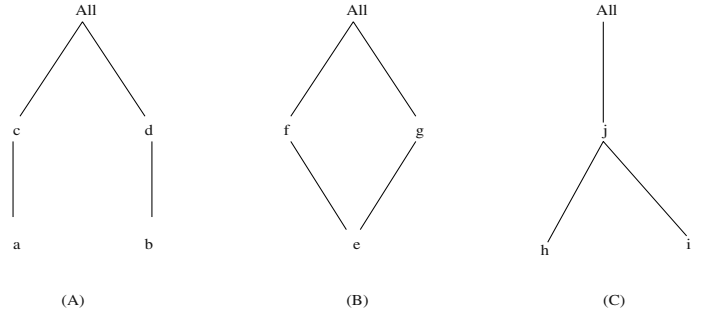


Fig. 4. Three hierarchy schemas.

It is not difficult to check that if $D \cong D'$ then $D \equiv_h D'$. We end this section by showing that it is straightforward to show that the converse also holds for balanced schemas.

A dimension instance d is *exact* if d is bijective. It is easily verified that all canonical dimension schemas have an exact dimension instance.

Theorem 1 (h-Equivalence of Balanced Schemas). *Two balanced dimension schemas $D = (H, \Sigma)$ and $D' = (H', \Sigma')$ are h-equivalent if and only if H and H' are (graph) isomorphic.*

Proof. (Sketch.) One direction is obvious.

Assume that $D \equiv_h D'$ via f. Consider an exact dimension d of (H, Σ). Then as graphs $H \cong \mathrm{dom}(d) \cong \mathrm{dom}(f(d))$. Now, because D' is balanced there is a (graph) monomorphism $\mu : \mathrm{dom}(f(d)) \to H'$ with $\mu(\mathtt{all}) = \mathtt{All}$ (if $\mu(v) = \mu(w)$ for $v \neq w$, the source of $\mathrm{dom}(f(d))$ would have two ancestors in the same category, violating condition 3 of Definition 3.) Hence there is a monomorphism $H \to H'$. By the same argument on the reverse direction, there is a monomorphism $H' \to H$. Hence because H, H' are finite graphs, $H \cong H'$. □

4 Hierarchical Equivalence of Canonical Schemas

This section extends the results of Theorem 1 to canonical dimensions. The importance of this result is twofold: (1) The notion of h-equivalence has a simple and intuitive characterization as graph isomorphism in canonical schemas (this result is stated in Theorem 2 in this section). (2) From Theorem 2, it follows that canonical schemas are strictly more expressive than balanced schemas (because given a canonical and not balanced schema there is no balanced schema isomorphic to it.) So we have now a formal argument that justifies the introduction of canonical schemas for OLAP modeling.

First, observe that the argument in the proof of Theorem 1 does not necessarily work for canonical schemas (there could be no injective μ).

oarieli@mta.ac.il

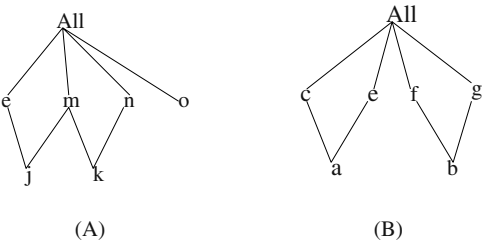


Fig. 5. Two hierarchy schemas.

Example 5. Let D and D' be the dimension schemas of figures 5 (A) and 5 (B), respectively. We define μ as in the proof of Theorem 1. Here, however, μ is not necessarily bijective. In particular, it could be the case that $h(d)$, where d is the exact dimension of D, is a dimension whose non-empty categories are a, c, e, and `All`. And therefore, μ could not be injective.

The following is the main result of the section. We need the following notation: a dimension d is complete with respect to a subgraph H' of its hierarchy schema if $\text{ran}(d) = H'$.

Theorem 2 (h-Equivalence of Canonical Schemas). *Let $D = (H, \Sigma)$ and $D' = (H', \Sigma')$ be two canonical schemas. Then, $D \equiv_h D'$ if and only if H is (graph) isomorphic to H'.*

Proof. (Sketch.) Let us sketch the non-trivial direction of the proof. Let $H = (C, \nearrow)$ and $H' = (C', \nearrow')$ and $f : I(D) \to I(D')$ be the bijection given by $\equiv_h$.

(*) Let $d_1 : (M, <) \to H$ be an exact dimension of D (hence $H \cong (M, <)$). Let $f(d_1) : (M, <) \to H'$ be the image of d_1 under f (by hypothesis $f(d_1)$ has the same domain as d_1). Let d'_1 be an exact dimension of $f(d_1)(M)$. Let d_2 be an exact dimension of $f^{-1}(d'_1)$. Continue this process until $H_1 = \text{Im}(d_i) \cong \text{Im}(d'_i) = H'_1$. Denote by μ_1 this isomorphism. Note that H_1 is well defined because the process terminates by a graph theoretic argument.

For each dimension instance $d : M_1 \to H$ with $d(M_1) = H_1$ do: Redefine f by performing the following operations: $y := f(d)$; $f(d) := (\mu_1 \circ d)$; $ff^{-1}(\mu_1 \circ d) := y$. Recall from Section 2 that an instance d takes its domain from a possibly infinite set $\mathbf{M}$. Here we assume that the set $\mathbf{M}$ is finite, hence the loop ends. The extension to the infinite case is straightforward. It is easily verified that at the end of this process we will have that for all complete d of H_1, it holds that $f(d) = (\mu_1 \circ d)$. Call f_1 this new f.

Now we repeat the whole process starting from (*) with f_1. This process generates a H_2, H'_2 and a new f_2.

Observe that $H_2 \neq H_1$, because otherwise there must be a complete dimension d of H_1 which is not mapped to the complete dimension $(\mu_1 \circ d)$ via f_1.

By repeating this process we generate a series $(H_1, H'_1, f_1), (H_2, H'_2, f_2), \ldots$ This series has the property $H_i \neq H_j$ for $i \neq j$ by an argument similar to the case $i = 1$. Finally just note that this series must be infinite, but there are only finitely many subgraphs of each hierarchy schema, a contradiction. □

oarieli@mta.ac.il

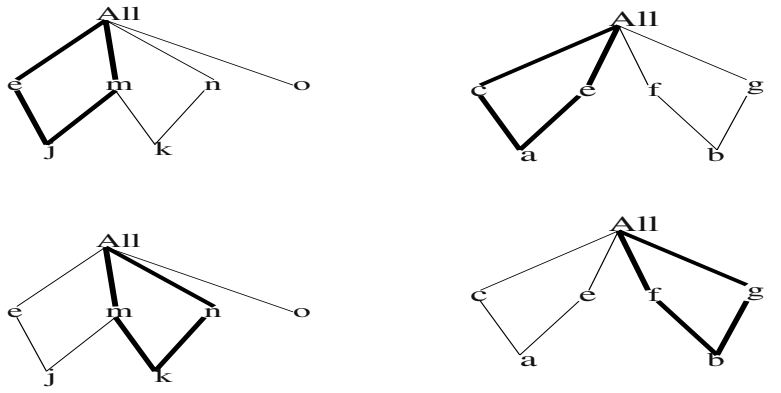


Fig. 6. A series of matchings illustrating proof of Theorem 2

The following examples illustrates the main idea of the previous proof.

Example 6. Let D and D' be the dimension schemas of Figures 5(A) and 5(B). Clearly they are not graph isomorphic. Assume that $D \equiv_h D'$ via an instance mapping f. Figure 6 depicts, on the top, the triple (H_1, H_1', f_1), and in the bottom (H_2, H_2', f_2), in a possible sequence generated in the proof for D and D'. The map f_1 sends the complete dimension of the subschema underlined to the one underlined in H_1'. Similarly for f_2. This property forces the schema H_2 (resp. H_2') to be different from H_1 (resp. H_1'). This series is infinite, but it can be checked now that there is no next triple (H_3, H_3', f_3), yielding a contradiction. Hence $D \not\equiv_h D'$.

5 Hierarchical Equivalence of Dimension Schemas

In this section we present a characterization of hierarchical equivalence for dimension schemas, which yields an algorithm for testing hierarchical equivalence. The characterization will be based on another notion of equivalence, which is defined in terms of mappings between finite sets of minimal dimensions conveyed by the schemas called frozen dimensions.

5.1 Frozen Equivalence

We introduce a notion of equivalence, *frozen equivalence*, defined in terms of injective mappings between special kinds of dimension instances, called frozen. Intuitively, a *frozen dimension* is a minimal dimension conveyed by a dimension schema. Frozen dimensions were introduced in previous work [HM02] to test implication of dimension constraints. In order to test whether a dimension schema satisfies a dimension constraint α, we only need to check α in each frozen dimension of the schema, which yields a finite set of tests (exponential in the size

oarieli@mta.ac.il

of the schema). In this section we use the notion of frozen dimension for giving an algorithmic version of h-equivalence.

Let $D = (H, \Sigma)$ be a dimension schema, we denote by $\texttt{Const}_D(c)$ the set of constants k that occur in atoms of the form $\langle c_i, .., c = k\rangle$ in Σ.

Definition 10 (Frozen Dimension). *Given a dimension schema D and $c \in C$, a* frozen dimension *with root c is a dimension instance $d : (M, <) \to (C, <)$ of D such that:*

1. d is injective (i.e., each category has at most one member);

2. $d^{-1}(c)$ is a source of $(M, <)$;

There could be infinitely many frozen dimensions, but there are only finitely many up to isomorphism, where isomorphism is defined as follows: d is isomorphic to d' iff there exists a graph mapping $f : (M, <) \to (M', <')$ such that $d = d' \circ f$, and if $k \in \texttt{Const}_D(c_j)$ and $d(k) = c_j = d'(k)$, then $f(k) = k$.

From now one, we will consider frozen dimensions up to isomorphism. Given a dimension schema D and a category c of it, we denote by $\texttt{Frozen}(D, c)$ the set of frozen dimensions of D (up to isomorphism) with root c, and by $\texttt{Frozen}(D)$ the union of all $\texttt{Frozen}(D, c)$ for all categories c of D.

Example 7. Figure 7 (top) shows three subgraphs of the hierarchy schema of the schema `productA` (given in Example 2). Each subgraph is induced by non-empty edges of a frozen dimension of `productA` with root *Product*. Intuitively, the frozen dimensions show the different structures that are mixed in the schema `productA`. Recall that frozen dimensions are dimension instances, but due to lack of space we do not show them directly.

The following notion of equivalence compares the information capacity of two schemas D and D' based on the frozen dimension they convey. To compare D and D' we establish a correspondence Ω between their sets of categories and then check that it induces a bijective relation between their frozen dimensions.

Definition 11 (Frozen Equivalence). *Let $D = (H, \Sigma)$ and $D' = (H', \Sigma')$ be dimension schemas, where $H = (C, \nearrow)$ and $H' = (C', \nearrow')$:*

Two frozen dimensions $d \in \texttt{Frozen}(D)$ and $d' \in \texttt{Frozen}(D')$ are isomorphic iff there exists a graph isomorphism $f : (M, <) \to (M', <')$ such that if $c \in C$ and $k \in \texttt{Const}_D(c)$, then $f(k) = k$. Notice that f induces a isomorphism $\bar{f} : Im(d) \to Im(d')$.

Let $\Omega \subseteq C \times C'$ be a category correspondence. An Ω-frozen relation $\mathcal{F}_\Omega \subseteq \texttt{Frozen}(D) \times \texttt{Frozen}(D')$ is defined as the set of pairs (d, d') related by an isomorphism $f : d \to d'$ such that $\bar{f} \subseteq \Omega$.

Two schemas D and D' are frozen equivalent (in the sequel f-equivalent) if there exists a bijective Ω-frozen relation from $\texttt{Frozen}(D)$ to $\texttt{Frozen}(D')$.

Example 8. Consider the dimension schemas `productA` and `productB`, and a category correspondence Ω between them having the pairs of categories (*Department*, *Department&AsisBranch*), (*Branch*, *Department&AsisBranch*) and a pair (c, c) for each category c in both schemas. The Ω-frozen relation between the sets $\texttt{Frozen}(\texttt{productA})$ and $\texttt{Frozen}(\texttt{productB})$ is showed in Figure 7. (the figure only shows the frozen dimensions with root Product.)

oarieli@mta.ac.il

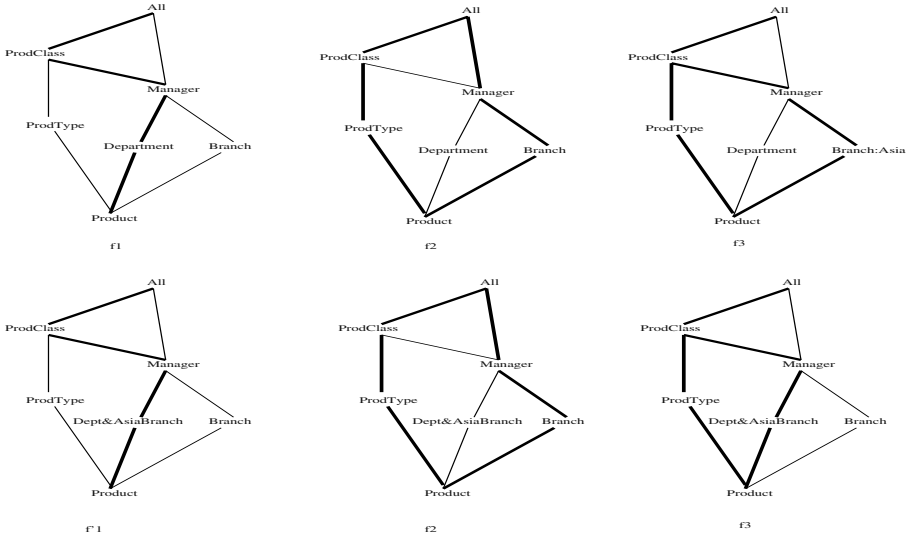


Fig. 7. Frozen Relation between the frozen dimension of `productA` and `productB`.

Proposition 1 (f-Equivalence implies h-Equivalence). *Let D and D' be two dimension schemas. If D and D' are f-equivalent, then D and D' are h-equivalent.*

The proof of this Proposition builds a bijective mapping $f : I(D) \to I(D')$ using the bijective frozen relation. Intuitively, if there is a 1-1 frozen relation (induced by some category correspondence between the schemas) then we can also define a 1-1 instance mapping between the instances of schemas. In Section 5.2 we will state the converse and sketch its proof.

5.2 h-Equivalence and f-Equivalence

In this section we show that h-equivalence implies f-equivalence. This result along with Proposition 1 shows that f-equivalence characterizes h-equivalence.

Firstly, we will introduce *frozen schemas*, dimension schemas that are normal forms, in the sense that every dimension schema is h-equivalent to a frozen schema.

Definition 12 (Frozen Schema). *A* frozen schema *is a dimension schema D such that each category c in D has a unique frozen dimension d and $Im(d)$ is exactly the upgraph of c.*

The following are some basic properties of frozen schemas: their dimension instances are homogeneous; they do not have shortcuts; and they subsume canonical schemas.

Example 9. The dimension schema `productC` given in Example 2 is a frozen schema, because its constraints cause each category c to have a single frozen

oarieli@mta.ac.il

dimension with root c. The category *AsiaBranch* is the only category whose frozen dimension has a constant (*Asia*). Schemas `productA` and `productB` are not frozen schemas since they convey several frozen dimensions with the category *Product* as root.

Next, we show that testing h-equivalence of frozen schemas defined over the same set of constants reduces to testing whether the schemas are isomorphic. This result generalizes Theorem 2 because canonical schemas are frozen schemas.

Note that two isomorphic frozen dimension must have the same set of constants. We will say that two frozen schemas are normalized w.r.t. a set of constants if for all $c \in C$ and $c' \in C'$ it holds $\texttt{Const}_D(c) = \texttt{Const}_{D'}(c')$, i.e., their equality atoms mention the same constants for each category.

Theorem 3 (h-Equivalence of Frozen Schemas). *Let D and D' be two normalized frozen schemas. Then D and D' are h-equivalent iff they are isomorphic (i.e., $D \equiv_h D'$ iff $D \cong D'$).*

The proof is a generalization of the proof of Theorem 2. This theorem also shows that dimension schemas are more expressive than canonical schemas because some frozen schemas are not isomorphic to any canonical schema. That is, there are dimension schemas for which there are no h-equivalent canonical schemas.

Finally, we prove the main result of this section.

Theorem 4 (h-Equivalence of dimension Schemas). *Let D and D' be two normalized frozen dimension schemas. Then D and D' are f -equivalent iff they are h-equivalent.*

Proof. (Sketch.) One direction is Proposition 1.

So assume that D and D' are h-equivalent. First define a schema transformation that takes D and produces a frozen schema D_f h-equivalent to D. The transformation works as follows: (1) Compute the frozen dimensions of D using the DIMSAT algorithm presented in previous work [HM02]; (2) Reverse the graph $(C, \nearrow')$ and do a topological sort of the resulting graph. (3) Follow the topological sort, and for each category c with more than one frozen dimension, split c into $c_1, \ldots, c_n$ (preserving adjacent edges). Add constraints to the schema in order to have a single frozen dimension in each category $c_1, \ldots, c_n$. This process yields a new dimension schema with a single frozen dimension in each category; (4) For each category c_j of the schema delete adjacent edges that do not match the frozen dimension.

Each split in step (3) induces induces the following category correspondence between the hierarchy schemas before and after the split: (c, c_j) for all $1 \leq j \leq n$, and for the remaining categories c' that appear in both hierarchy schemas we have (c, c'). It is not difficult to verify that this category correspondence induces a bijective frozen relation between the old and the new schema. By composing these frozen relations we get a bijective frozen relation between D and D_f.

In the same manner, we built a frozen schema D'_f and a bijective frozen relation between D' and D'_f. Hence, we have bijective frozen relations $D \to D_f$

oarieli@mta.ac.il

and $D' \to D'_f$. Also, from Theorem 3 we know that $D_f \cong D'_f$ via some μ. From μ we can derive a bijective frozen relation between D_f and D'_f. Composing these relations we derive the statement of the theorem. □

Example 10. The bijective frozen relation of Figure 7 shows that the schemas `productA` and `productB` of Example 2 are h-equivalent. By a simmilar procedure it can be easily verified that schema `productC` is h-equivalent to schemas `productA` and `productB`.

5.3 Transforming Dimension Schemas into Homogeneous Schemas

From the proof of Theorem 4 it follows that any dimension schema can be transformed into a hierarchically equivalent homogeneous schema. In Figure 8 we sketch an algorithm to perform such a transformation.

The algorithm outputs dimension schemas having the constraints that state the homogeneity condition. The schemas may also have additional constraints with equality atoms. Path atoms other than the ones that state the homogeneity condition are irrelevant for the resulting schemas because a path atom is either $\bot$ or $\top$ in every instance of a homogeneous schema. In Line 2 the algorithm computes the set of frozen dimensions of D. In Line 5, for each category c of D and subset S of categories directly above c, the algorithm adds to H a new category $\texttt{CatName}(n)$ (the function $\texttt{CatName}(n)$ returns a name for a new category when a integer n is given) connected to the categories that are connected to c; then, the set of frozen dimensions F is updated in order to keep them consistent with the fact that the new category $\texttt{CatName}(n)$ represents the members that have parents only in categories in S. In Line 10 the algorithm does a traversal of the set F deleting empty edges in the hierarchy schema, and adding to Σ' equality atoms associated to the constants that appear in F. In this step the algorithm also adds the constraints of the form $\langle c, c' \rangle$ (homogeneity constraints) for each edge (c, c') in the resulting hierarchy schema.

In previous work [HM02] we provided an algorithm to compute the set of frozen dimensions of a schema in exponential time on the size of the schema. Thus, we can prove:

Proposition 2. *Assume the set of frozen dimensions of a schema are computed, then the algorithm of Figure 8 runs in time $O(NF2^N)$, where N is the size of the hierarchy schema, and F is the number of frozen dimensions.*

5.4 Algorithmic Aspects of Testing Hierarchical Equivalence

From the proof of Theorem 4, we can derive an algorithm for testing h-equivalence and prove that this problem is decidable. The naive application of the procedure in the proof yields a double exponential time algorithm. In fact, we can test whether $D \equiv_h D'$ in the following two steps: (1) apply the transformation in step (4) in the proof to transform D into D_f and D' into D'_f; and (2) test whether H_f is graph isomorphic to H'_f, where H_f (resp. H'_f) is the hierarchy schema of D_f (resp. D'_f).

oarieli@mta.ac.il

Input: A dimension schema $D = (H, \Sigma)$
Output: A homogeneous dimension schema D', such that D is h-equivalent to D'
(1) Let C_{ini} be the set of categories of D, $\Sigma_H := \emptyset$; $T := \emptyset$; $n := 0$
(2) Compute the set of frozen dimensions F of D
(3) For every category $c \in C_{ini}$ do
(4) For every non-empty set S of categories connected from c do
(5) Split c into $\texttt{CatName}(n)$ and c in H
(6) For each frozen dimension f of F, if $f \models \bigwedge_{c_i \in S} \langle c, c_i \rangle \wedge \bigwedge_{\{c_j : c \nearrow c_j\} \setminus S} \langle c, c_j \rangle$ then rename c with $\texttt{CatName}(n)$ in f;
(7) $n := n + 1$
(8) EndFor
(9) EndFor
(10) Scan the set F deleting the empty edges from H and adding equality and homogeneity constraints to Σ'
(11) Return (H, Σ')
End

Fig. 8. Algorithm that transforms a dimension schema into an h-equivalent homogeneous schema.

The number of categories in the frozen schemas is in $O(n2^n K)$, where n is the number of categories and K is the number of constants mentioned in the schema. This bound is the order of the number of splits used in each transformation. Essentially, we may have as many categories in the resulting schema as frozen dimensions in the original schema. Since the size of D_f (resp. D'_f) is exponential in the size of the initial schema D (resp. D') we get the stated bound due to the test in 2.[1]

The following result shows that the problem is hard.

Theorem 5 (Testing h-Equiv.). *Testing whether two dimension schemas D and D' are h-equivalent is co-NP hard.*

The proof is a reduction of VALIDITY (given a proposition P, is P satisfied by all truth assignments?) to this problem.

We end this section by sketching an exponential time algorithm for testing h-equivalence: (1) compute the frozen dimensions of D and D'; and (2) for every binary relation between categories, test whether it induces a bijective frozen mapping.

Step 1 can be done in exponential time on the size of the schema. (See [HM02] for detailed bounds.) The number of binary relations between categories we need to test in Step 2 is $O(2^{n^2})$. For each such relation, we have to compute the induced frozen relation R, i.e. we need to test for each pair of frozen dimensions $d \in \texttt{Frozen}(D)$ and $d' \in \texttt{Frozen}(D')$ whether $(d, d') \in R$. This test can be done

[1] DAG isomorphism is graph isomorphism complete. Recall that the "exact" complexity of deciding whether two graphs are isomorphic is still not known. The problem has neither been proved to be NP complete nor in P.

oarieli@mta.ac.il

in 2^{n^2} operations of $O(n)$ steps each, since we need to check at most 2^{n^2} possible isomorphisms between d and d'. Also, we have to perform one test for each pair of frozen dimensions d and d'. Since the number of frozen dimensions of a given schema is exponential in the size of the schema, Step 2 can be accomplished in time exponential on the size of the schemas.

6 Conclusion and Further Work

In this paper we have presented a series of results that give conceptual insights into the problem of modeling OLAP dimension schemas. In particular, our framework: allowed us to compare different classes of dimension schemas introduced in a variety of OLAP models; and provides a formal basis to further research on schema restructuring in OLAP warehouses.

Dimension schemas enriched with dimension constraints give users flexibility to choose among several options the best suited for the application at hand. Further work needed to turn this flexibility into practical OLAP applications includes the definition of normal forms, restructuring operators, and implementation issues.

Acknowledgments. We thank Alberto Mendelzon for fruitful suggestions on early versions of this work. This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Mideplan, Chile.

References

[Alb00] J. Albert. Theoretical foundations of schema restructuring in heterogeneous multidatabase systems. In *Proceedings of the ACM Conference on Information and Knowledge Management*, Washington, DC, USA, 2000.

[AV97] S. Abiteboul and V. Vianu. Regular path queries with path constraints. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, Tucson, Arizona, USA, 1997.

[BFS98] P. Buneman, W. Fan, and Weinstein S. Path constraints on semistructured and structured data. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, Seattle, Washington, USA, 1998.

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. In *ACM SIGMOD Record 26(1)*, March 1997.

[CT97] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Proceedings of the 6th International Workshop on Database Programming Languages*, East Park, Colorado, USA, 1997.

[Gol81] B. A. Goldstein. Constraints on null values in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases*, Cannes, France, 1981.

[HG03] C. Hurtado and C. Gutierrez. Equivalence of OLAP dimension schemas (extended version). In *Technical Report, Departamento de Ciencias de la Computación, Universidad de Chile, TR/DCC-2003-7*, 2003.

oarieli@mta.ac.il

[HGM03] C. Hurtado, C. Gutiérrez, and A. Mendelzon. Capturing summarizability with integrity constraints in OLAP. In *Technical Report, Departamento de Ciencias de la Computación, Universidad de Chile, TR/DCC-2003-6*, 2003.

[HLV00] B. Huseman, J. Lechtenborger, and G. Vossen. Conceptual data warehouse design. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW)*, Stockholm, Sweden, 2000.

[HM02] C. Hurtado and A. Mendelzon. OLAP dimension constraints. In *Proc. PODS 2002*, Madison, USA, 2002.

[HMV99] C. Hurtado, A. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. In *Proceedings of the 15th IEEE International Conference on Data Engineering (ICDE)*, Sydney, Australia, 1999.

[Hul86] R. Hull. Relative information capacity of simple relational database schemata. In *SIAM Journal of Computing 15(3):865-886*, 1986.

[Hur02] C. Hurtado. Structurally heterogeneous OLAP dimensions. In *Ph.D. Thesis, Department of Computer Science, University of Toronto*, 2002.

[JLS99] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *Proc. of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, UK, 1999.

[LAW98] W. Lehner, H. Albrecht, and H. Wedekind. Multidimensional normal forms. In *Proceedings of the 10th Statistical and Scientific Database Management Conference*, Capri, Italy., 1998.

[MIR94] R. Miller, Y. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. In *Information Systems, vol. 19, no.1*, 1994.

[PJE99] T. B. Pedersen, C. S. Jensen, and Dyreson C. E. Extending practical preaggregation in on-line analytical processing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, 1999.

[VL00] Millist W. Vincent. and Mark Levene. Restructuring partitioned normal form relations without information loss. *SIAM Journal on Computing*, 29(5):1550–1567, 2000.

oarieli@mta.ac.il

A New Approach to Belief Modeling

V.N. Huynh[1], Y. Nakamori[1], T. Murai[2], and T.B. Ho[1]

[1] School of Knowledge Science
Japan Advanced Institute of Science and Technology
Tatsunokuchi, Ishikawa, 923-1292, JAPAN
{huynh,nakamori,bao}@jaist.ac.jp

[2] Graduate School of Engineering
Hokkaido University
Kita 13, Nishi 8, Kita-ku, Sapporo 060-8628, Japan
murahiko@main.eng.hokudai.ac.jp

Abstract. It has been shown that, despite the differences in approach and interpretation, all belief function based models without the so-called *dynamic component* lead essentially to mathematically equivalent theories – at least in the finite case. In this paper, we first argue that at the logical level these models seem to share a common formal framework and various interpretations just come at the epistemic level. We then introduce a framework for belief modeling formally based on Dempster's structure with adopting Smets' view of the origin of beliefs. It is shown that the proposed model is more general than previous models, and may provide a suitable unified framework for belief modeling.

Keywords: Transferable belief model, Uncertainty, Dempster–Shafer theory, Propagable belief model.

1 Introduction

Dealing with uncertainty is a fundamental and unavoidable issue in AI researches. Undoubtedly, the Bayesian approach is the most widely-used approach to dealing with uncertainty. Although the Bayesian approach is strongly supported by relying on well-established techniques from probability theory as well as some philosophical justification, it has been widely criticized in the literature. So far numerous other approaches to dealing with uncertainty have been proposed, including Dempster-Shafer theory [2,22], the transferable belief model [24, 28], the probability of modal propositions [21], various nonstandard and fuzzy logics [16,10,32], and the context model [7], among others. Of particular interest to us in this paper is based on the Dempster-Shafer-Smets model[1].

From a mathematical point of view, a belief function can be treated as a mathematical object satisfying a certain set of axioms. Especially, the axioms for

[1] This name is used in [7] to reflect Smets' "non-probabilistic" view of using belief functions (including Dempster's rule of conditioning and Dempster's rule of combination) to model someone's belief.

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 196–212, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

belief functions can be viewed as a weaker form of the Kolmogorov axioms that characterize probability functions. Under such a view, a number of authors have tried to characterize a belief function as a generalized probability function [5,6] or in terms of probability functions [2,3,23]. On the other hand, belief functions have been also used to model someone's belief originated back to Shafer [22]. In belief modeling using belief functions, there are various views, even contrast, of the origin of beliefs. These have resulted in so many various interpretations of Dempster-Shafer theory and, at the same time, opened to criticism [29]. This paper does not aim at being a deal of debate regarding the existing approaches to belief modeling, also not presenting another interpretation of belief functions. Our main concern is on belief modeling itself. To this end, we adopt Smets' view of the origin of beliefs in the transferable belief model, inasmuch as it is not only based on a well-established axiomatic justification, but also supported by practical basis when someone intends to model subjective, personal beliefs. For example, in a medical diagnostic situation, it is easier and realizable for You, the doctor, to give basic belief masses on subsets of symptoms that may cause the unknown disease rather than to give (subjective) probabilities on single symptoms, even though such probabilities may exist[2]. On the other hand, we are highly motivated by the fact that the notion of a multi-valued mapping may be a good mathematical tool for representing human beings' *cause-and-effect* view of reality. Thus our approach is based on Dempster's structure, but according to Kohlas and Monney's view of the multi-valued mapping [13].

In the next section we will briefly present necessary notions from the Dempster-Shafer theory of evidence. Some belief function based models are recalled and analyzed in Section 3. We would like to emphasize that the model introduced in this paper should not be considered as a formally generalization of previous models, even though it may be. Thus not all interpretations of belief functions are analyzed here (see [29] for the details), but only models that we have been guided by our purpose are mentioned. A full description of the model for beliefs representation can be found in [22]. Other models can be found in, e.g. [21,20,15] for the modal logic based interpretation; [17] for the random set based interpretation; [7,11] for the context model. In Section 4 we introduce the so-called propagable belief model, and conditioning as belief revision with certain evidence versus the one with uncertain evidence will be analyzed via the well-known *tree prisoners problem*. Finally, some concluding remarks and further work will be presented in Section 5.

2 Dempster-Shafer Theory of Evidence

We recall in this section necessary notions from the Dempster-Shafer theory of evidence (DS theory, for short). The theory aims at providing a mechanism for representing and reasoning with uncertain, imprecise and incomplete informa-

[2] Note that this does not exclude the possibility of using correct probabilities whenever available.

oarieli@mta.ac.il

tion. It is based on Dempster's original work [2] on the modeling of uncertainty in terms of upper and lower probabilities induced by a multi-valued mapping.

A multi-valued mapping Γ from space Ω into space Θ associates to each element ω of Ω a subset $\Gamma(\omega)$ of Θ. The domain of Γ, denoted by $\mathsf{Dom}(\Gamma)$, is defined by

$$\mathsf{Dom}(\Gamma) = \{\omega \in \Omega | \Gamma(\omega) \neq \emptyset\}.$$

From a multi-valued mapping Γ, a probability measure P on Ω can be propagated to Θ in such a way that for any subset T of Θ the lower and upper bounds of probabilities of T are defined as

$$P_*(T) = \frac{P(\Gamma_*(T))}{P(\mathsf{Dom}(\Gamma))} \tag{1}$$

$$P^*(T) = \frac{P(\Gamma^*(T))}{P(\mathsf{Dom}(\Gamma))} \tag{2}$$

where

$$\Gamma_*(T) = \{\omega \in \Omega | \omega \in \mathsf{Dom}(\Gamma) \wedge \Gamma(\omega) \subseteq T\},$$
$$\Gamma^*(T) = \{\omega \in \Omega | \Gamma(\omega) \cap T \neq \emptyset\}.$$

Clearly, P_*, P^* are well defined only when $P(\mathsf{Dom}(\Gamma)) \neq 0$.

Remark 2.1. The equations (1) and (2) can be represented in the terms of conditional probabilities as follows

$$P_*(T) = P(\Gamma_*(T)|\mathsf{Dom}(\Gamma)), \quad P^*(T) = P(\Gamma^*(T)|\mathsf{Dom}(\Gamma)) \tag{3}$$

This presentation suggests us the idea of a new interpretation of conditional beliefs presented in Section 4.

Furthermore, Dempster also observed that, in the case that Θ is finite, these lower and upper probabilities are completely determined by the quantities

$$P(\{\omega \in \Omega | \Gamma(\omega) = T\}), \text{ for } T \in 2^\Theta.$$

As such Dempster implicitly gave the prototype of a mass function also called *basic probability assignment*. Shafer's contribution has been to explicitly define the basic probability assignment and to use it to represent evidence directly. Simultaneously, Shafer has reinterpreted Dempster's lower and upper probabilities as degrees of belief and plausibility respectively, and abandoned the idea that they arise as lower and upper bounds over classes of Bayesian probabilities [22].

Formally, the definitions of these measures are given as follows:

1. A function $bel : 2^\Theta \to [0, 1]$ is called a *belief measure* over Θ if
   B1. $bel(\emptyset) = 0, bel(\Theta) = 1$
   B2. For any finite family $\{A_i\}_{i=1}^n$ in 2^Θ,

$$bel(\bigcup_{i=1}^{n} A_i) \geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} bel(\bigcap_{i \in I} A_i)$$

oarieli@mta.ac.il

2. A function $pl : 2^{\Theta} \to [0,1]$ is called a *plausibility measure* if
   P1. $pl(\emptyset) = 0, pl(\Theta) = 1$
   P2. For any finite family $\{A_i\}_{i=1}^{n}$ in 2^{Θ},

$$pl(\bigcap_{i=1}^{n} A_i) \leq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} pl(\bigcup_{i \in I} A_i)$$

It should be noted that belief and plausibility measures form a dual pair, namely

$$pl(A) = 1 - bel(\overline{A}), \text{ for any } A \in 2^{\Theta}$$

In the case of a finite universe S, a function $m : 2^{\Theta} \to [0,1]$ is called a *basic probability assignment* if $m(\emptyset) = 0$ and

$$\sum_{A \in 2^{\Theta}} m(A) = 1$$

A subset $A \in 2^{\Theta}$ with $m(A) > 0$ is called a *focal element* of m. The difference between $m(A)$ and $bel(A)$ is that while $m(A)$ is our belief committed to the subset A excluding any of its proper subsets, $bel(A)$ is our degree of belief in A as well as all of its subsets. Consequently, $pl(A)$ represents the degree to which the evidence fails to refute A. Furthermore, the belief and plausibility measures are in an one-to-one correspondence with basic probability assignments. Namely, given a basic probability assignment m, the corresponding belief measure *bel* and its dual plausibility measure *pl* are determined by

$$bel(A) = \sum_{\emptyset \neq B \subseteq A} m(B)$$

$$pl(A) = \sum_{B \cap A \neq \emptyset} m(B)$$

Conversely, given a belief measure *bel*, the corresponding basic probability assignment m is determined via Möbius inversion as follows

$$m(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} bel(B)$$

In the next section we will briefly present several various interpretations of the DS model, namely Kohlas and Monney's hint model [14], Fagin and Halpern's model [5] and Smets' transferable belief model [28]. In this paper we ourselves confine the consideration to only the finite structures.

3 Belief Function Based Models

Since Shafer introduced the model in the seminal work "*A Mathematical Theory of Evidence*" [22], many interpretations of it have been proposed. According to Smets [29], any model for belief has at least two components: one *static* that describes our state of belief, and the other *dynamic* that explains how to update our belief given new pieces of information. It has been clear that by restricting to the static component, various models for belief, despite the differences in approach and interpretation, lead essentially to mathematically equivalent forms.

oarieli@mta.ac.il

3.1 The Hint Model

The hint model proposed by Kohlas [12] and developed further by Kohlas and Monney [13,14] begins with Dempster's original structure $(\Omega, P, \Gamma, \Theta)$ where Ω and Θ are two sets, P is a probability measure on Ω and Γ is a multi-valued mapping from Ω into Θ.

The authors assume a certain question, whose answer is unknown. The set Θ called the *frame of discernment* is the set of possible answers to the question. One and only element of Θ is the correct answer but unknown. Ω is interpreted as the set of *possible interpretations* allowed from the light of the available information. Exactly one of the elements $\omega \in \Omega$ must be the correct interpretation, but it is unknown which one. Furthermore, the assumption that not all possible interpretations are equally likely induces the known probability measure P on Ω. In the simplest case, one can assume that if ω is the correct interpretation, then the correct answer θ must be within some nonempty subset $\Gamma(\omega)$ of Θ, the *focal set* of the interpretation. Alternately, for any possible interpretation ω, the family $\mathcal{S}(\omega)$ of the subsets of Θ (considered as propositions) **implied** by ω can be considered. $\mathcal{S}(\omega)$ called a *filter* is simply the family of supersets of the focal set $\Gamma(\omega)$ and has the following properties:

(1) $H \in \mathcal{S}(\omega)$ and $H \subseteq H'$ imply $H' \in \mathcal{S}(\omega)$.
(2) $H_1, H_2 \in \mathcal{S}(\omega)$ imply $H_1 \cap H_2 \in \mathcal{S}(\omega)$.
(3) Θ belongs to $\mathcal{S}(\omega)$, $\emptyset$ does not belong to $\mathcal{S}(\omega)$.

Furthermore, one can also look at the family $\mathcal{P}(\omega)$ of the propositions which are **possible** under ω. That is, a subset H of Θ is considered as possible if H does have a nonempty intersection with the focal set $\Gamma(\omega)$. The family $\mathcal{P}(\omega)$ has the following properties:

(1') $H \in \mathcal{P}(\omega)$ and $H \subseteq H'$ imply $H' \in \mathcal{P}(\omega)$.
(2') $H_1, H_2 \in \mathcal{P}(\omega)$ imply $H_1 \cup H_2 \in \mathcal{P}(\omega)$.
(3') Θ belongs to $\mathcal{P}(\omega)$, $\emptyset$ does not belong to $\mathcal{P}(\omega)$.

Under such an analysis, the quadruple $\mathcal{H} = (\Omega, P, \Gamma, \Theta)$ is called a **hint**.

Now if a proposition $H \subseteq \Theta$ is fixed as a hypothesis about the correct answer, then this hypothesis should be judged in the light of a hint $\mathcal{H}$. That is, one can look at the subsets of interpretations under which H is implied, $u(H)$, or possible, $v(H)$

$$\begin{aligned} u(H) &= \{\omega \in \Omega | H \in \mathcal{S}(\omega)\} \\ v(H) &= \{\omega \in \Omega | H \in \mathcal{P}(\omega)\} \end{aligned} \tag{4}$$

Then the *degree of credibility* (or *support*), denoted by $sp(H)$, and the *degree of plausibility*, denoted by $pl(H)$ are defined as follows

$$\begin{aligned} sp(H) &= P(u(H)) \\ pl(H) &= P(v(H)) \end{aligned} \tag{5}$$

As such the hint model is based on Dempster's original approach and in this model degrees of supports (or equivalently, beliefs) are deduced from a filter-valued mapping and a probability measure on the space of possible interpretations.

oarieli@mta.ac.il

Remark 3.1. **(i)** In the hint model one may implicitly assume a propositional language $\mathcal{L}_\Theta$ that is derived from the question of concern and is semantically interpreted by the Boolean algebra of 2^Θ. The filter-valued mapping induced from Γ plays an important role in forming credibility in the light of a hint. Thus, at the logical level a hint may be seen as a quadruple $(\Omega, \Gamma, \Theta, \mathcal{L}_\Theta)$.

(ii) In our opinion, the assumption "not all possible interpretations are equally likely" is not always available in general, once it is available it should be considered as the supplemental information and then the probability measure P on Ω is added to the hint to quantify degrees of credibility in the light of the hint. Furthermore, although a probability function is assumed on Ω, the hint model does not explicitly assume there is a probability function on Θ as upper and lower probabilities model does. Thus the hint model may be considered as a logical based interpretation associated with supplemental probabilistic information of the DS model.

3.2 Fagin and Halpern's Model

In [5] Fagin and Halpern introduced a new probabilistic approach to dealing with uncertainty by using the standard mathematical notions of *inner measure* and *outer measure* induced by the probability measure [8]. Interestingly, inner measures induced by probability measures turn out to correspond in a precise sense to DS belief functions. The model is interpreted as follows.

Let $\Phi = \{p_1, \ldots, p_n\}$ be a finite set of primitive propositions thought of as corresponding to basic events concerning with the situation we want to reason about. The set $\mathcal{L}(\Phi)$ of *propositional formulas* is the closure of Φ under the Boolean operations $\wedge$ and $\neg$. For convenience we assume also that there is a special formula *true*, and we abbreviate $\neg true$ by $false$. To get mutually exclusive events, we can consider all the formulas of the form $p'_1 \wedge \ldots p'_n$ called *atoms*[3], where p'_i is either p_i or $\neg p_i$. Let At denote the set of atoms over Φ.

In Nilsson's probabilistic logic [16], a probability distribution is assumed on At. Then the *probabilistic truth value* of a formula φ can be computed by using the finite additivity property of the probability measure and the equivalent representation of the formula φ as a disjunction of atoms. This formally forms a probability space of the form $(At, 2^{At}, P)$[4] called a *Nilsson structure*. Given a Nilsson structure $N = (At, 2^{At}, P)$ and a formula φ, let $W_N(\varphi)$ denote the probabilistic truth value (or shortly, *weight*) of φ in N, which is defined to be $P(At(\varphi))$, where $At(\varphi)$ is the set of atoms whose disjunction is equivalent to φ.

Fagin and Halpern have proposed a more general approach by taking a *probability structure* as a quadruple $(S, \mathcal{X}, P, \pi)$, where $(S, \mathcal{X}, P)$ is a probability space, π associates with each $s \in S$ a truth assignment $\pi(s) : \Phi \to \{\mathbf{true}, \mathbf{false}\}$. The equation $\pi(s)(p) = \mathbf{true}$ means that p is *true at s*. The set S is thought of as consisting of the possible states of the world. We can associate with each state s in S a unique atom describing the truth values of the primitive propositions in

[3] The terminology by Fagin and Halpern, also called *interpretations* in the logic literature.

[4] The notation P is used here instead of μ as in [5] to denote a probability measure.

oarieli@mta.ac.il

s. Further, there may be several states associated with the same atom. Using the usual rules of propositional logic we can easily extend $\pi(s)$ to a truth assignment on all formulas.

Given a probability structure $M = (S, \mathcal{X}, P, \pi)$, we now associate with each formula φ the set $\varphi^M = \{s \in S | \pi(s)(\varphi) = \mathbf{true}\}$ with assuming that $true^M = S$. If p^M is measurable for every primitive propositions $p \in \Phi$ then so is φ^M for every formula φ. In that case we say that M is a *measurable probability structure*. In general, we can not talk about the probabilistic truth value of a formula φ if φ^M is not measurable. In such a case, Fagin and Halpern proposed to use its inner measure and outer measure as these are defined for all subsets. Intuitively, the inner and outer measure provide lower and supper bounds on the probabilistic truth value of φ. Particularly, if φ^M is not measurable, we define $W_M(\varphi)$ to be the inner measure of φ in M as follows

$$W_M(\varphi) \stackrel{def}{=} P_*(\varphi^M) = \sup\{P(X) | X \subseteq \varphi^M, X \in \mathcal{X}\}$$

A proof given in [5] following from a more general result in [23] shows that P_* is indeed a belief measure. On the basis of the ideas above, the authors also developed a new notion of *conditional belief* which plays the same role for DS belief functions as conditional probability does for probability functions [6,9].

It is of interest that Fagin and Halpern's model can be viewed as a special case of Dempster's structure at least in the finite case as follows.

If S is a finite set, it is easy to see that $\mathcal{X}$ has a *basis*, i.e. a family $\mathcal{B}$ of nonempty and disjoint subsets of S such that every member of $\mathcal{X}$ is a union of members of $\mathcal{B}$. Furthermore, the basis $\mathcal{B}$ forms a partition of S, say $\mathcal{B} = \{B_1, \ldots, B_k\}$. We can now associate with each B_i a so-called *situation* t_i, which may be thought of as a realization of the possible states in B_i. Let T denote the set $\{t_1, \ldots, t_k\}$. In addition we define a probability distribution P_T on T as $P_T(t_i) = P(B_i)$, and a multi-valued mapping Γ from T into S by $\Gamma(t_i) = B_i$. Then it is easy to see that the Dempster structure (T, P_T, Γ, S) induces a belief function that coincides with Fagin and Halpern's proposal via the inner measure above.

3.3 The Transferable Belief Model

The transferable belief model (TBM, for short) introduced in [24,28] provides a model for the representation of quantified belief. This model is based on the assumption that beliefs manifest themselves at two mental levels: the *credal* level where beliefs are entertained and the *pignistic* level where beliefs are used to make decisions (from *credo*, I believe and *pignus*, a bet both in Latin). Especially, the TBM justifies the use of belief functions to model subjective, personal beliefs even in the cases where every probability concept is absent at the credal level. Once probabilities are defined everywhere the TBM is reduced to the Bayesian model [29]. The TBM is briefly described as follows.

Let $\mathcal{L}$ be a finite propositional language, and $W = \{w_1, w_2, \ldots, w_n\}$ be the set of possible worlds that correspond to the interpretations of $\mathcal{L}$. The set W is called the *frame of discernment*. Each proposition in $\mathcal{L}$ identifies a subset of W,

oarieli@mta.ac.il

and two propositions are logically equivalent iff they identify the same subset. Given a partition Π of W, we build the *Boolean algebra* $\mathcal{R}$ of subsets of W generated from Π. The elements of Π are called the *atoms* of $\mathcal{R}$, and the pair $(W, \mathcal{R})$ is called a *propositional space*.

Now assume that You is an ideal rational agent, and all beliefs entertained by You at time t about which world is the actual world ϖ are defined relative to a given *evidential corpus* ($\mathrm{EC}_t^{\mathrm{Y}}$). By the **Basic Assumption**, the TBM assume a *basic belief assignment* $m : \mathcal{R} \to [0, 1]$ with

$$\sum_{A \in \mathcal{R}} m(A) = 1, \quad m(\emptyset) = 0.$$

For $A \in \mathcal{R}$, $m(A)$ is a part of Your belief that supports A, i.e. that the actual world ϖ is in A, and that, due to the lack of information, does not support any strict subproposition of A. The difference with probability models here is that masses can be given to any proposition of $\mathcal{R}$ instead of only to atoms of $\mathcal{R}$. In the TBM, once some further evidence becomes available to You and implies that B is true, the mass $m(A)$ initially allocated to A is transferred to $A \cap B$. This transfer of belief in the TBM satisfies the so-called *Dempster rule of conditioning* and results in $m_B : \mathcal{R} \to [0, 1]$ with

$$m_B(A) = \begin{cases} c \sum\limits_{X \subseteq \overline{B}} m(A \cup X) & \text{for} \quad A \subseteq B, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$c = \frac{1}{1 - \sum\limits_{X \subseteq \overline{B}} m(X)}$$

Given a propositional space $(W, \mathcal{R})$ and a basic belief assignment m, the belief function $bel : \mathcal{R} \to [0, 1]$ is defined as usual by

$$bel(A) = \sum_{\mathcal{R} \ni X \subseteq A} m(X).$$

The triple $(W, \mathcal{R}, bel)$ is then called a *credibility space*.

At this juncture we can see that, given the evidence available on a situation You want to reason about, the TBM claims the existence of a belief function that describes Your credal state on the frame of discernment. Suppose now a *decision* must be made based on this credal state. As is well known [1] that decisions will be coherent if the underlying uncertainties can be described by a probability distribution defined on 2^W. Based on the *Generalized Insufficient Reason Principle* [28], the *pignistic probability distribution* derived from *bel* at the pignistic state via the so-called *pignistic transformation* is defined as follows

$$BetP(x) = \sum_{x \subseteq A \in \mathcal{R}} \frac{m(A)}{|A|} = \sum_{A \in \mathcal{R}} m(A) \frac{|x \cap A|}{|A|} \tag{6}$$

where x is an atom in $\mathcal{R}$ and $|A|$ is the number of atoms of $\mathcal{R}$ in A.

oarieli@mta.ac.il

Remark 3.2. If we denote $\mathcal{F}(x)$ the *principal filter* generated by an atom x in the Boolean algebra $\mathcal{R}$ [19], the pignistic probability distribution $BetP$ derived from m is represented as

$$BetP(x) = \sum_{A \in \mathcal{F}(x)} \frac{m(A)}{|A|} \tag{7}$$

This may show a logical relation implicitly behind the TBM and the hint model even though the primitive concepts of these two models are different. While in the hint model, the primitive concept is the hint from which degrees of supports are deduced, the TBM assume the degrees of belief as a primitive concept from which the pignistic probability function is derived. At the same time, as mentioned in [28] (page 200), the important concept in a propositional space $(W, \mathcal{R})$ is the algebra $\mathcal{R}$ (so is the partition Π), not the set of worlds W. Formally, similar as mentioned above in Fagin and Halpern's model, we can view the TBM in the terms of Dempster's structure without, however, reference to any probability concepts.

For the axiomatic justifications and more details on the TBM as well as its applications, the reader could be referred to, e.g. [4,25,27,29,30,31].

4 The Propagable Belief Model

In this section we introduce a model called *propagable belief model* (PBM, for short) that aims at presenting a new approach to modeling subjective, personal beliefs in the spirit of the TBM. Essentially, our model is based on Dempster's structure, except the assumption of a underlying probability distribution is not assumed. Instead of this we adopt the *basic assumption* as in the TBM.

4.1 The Model

The PBM concerns the same concepts as considered by previous models that are specified as follows.

Let $W = \{w_1, w_2, \ldots, w_n\}$ be the set of possible states of the world concerning a situation we want to reason about. We call W the frame of discernment and may think of elements of W as interpretations of a underlying propositional language, or possible answers to a given question, or the like. Practically, due to the complexity of the reasoning situation and/or lack of information, the information on W may be encoded into a nonempty *finite set of possible observations* $\mathcal{O}$. Each observation Ob in $\mathcal{O}$ can cover several possible states of the world, a subset $\Gamma(Ob)$ of W. In addition, we assume that all the available information allow us to allocate *belief masses* to subsets of the set of possible observations. For $O \in 2^{\mathcal{O}}$, $m_{\mathcal{O}}(O)$ is the belief degree that supports that the true state of the world ϖ is covered in the set of observations O. That is, due to lack of information, in some cases a belief mass is only assigned in a combined view of several observations but not any strict subset of these observations. For the discussion on the origin of the basic belief masses, we could be referred to [28].

oarieli@mta.ac.il

Example 4.1. Assume You, the detective, are dealing with a case of murder. You may determine a basic evidential structure consisting of W as the set of suspects who had potential to be the killer, $\mathcal{O}$ as the set of observed evidence in which each observed evidence supposes several suspects to be the killer, and $m_{\mathcal{O}} : 2^{\mathcal{O}} \to [0,1]$ as the basic belief assignment, where $m_{\mathcal{O}}(O)$, for $O \in 2^{\mathcal{O}}$, quantifies Your belief degree supporting that observed evidence in O constitute the murder.

Example 4.2. In a medical diagnostic situation, You, the doctor, may determine a basic evidential structure for diagnosis consisting of W as the set of possible diseases which the present patient may get, $\mathcal{O}$ as the set of observed symptoms from the patient in which, according to Your experience, each symptom may occur in several diseases, and $m_{\mathcal{O}} : 2^{\mathcal{O}} \to [0,1]$ as the basic belief assignment, where $m_{\mathcal{O}}(O)$, for $O \in 2^{\mathcal{O}}$, quantifies Your belief degree supporting that symptoms in O causes the unknown disease.

Formally, we define a *basic evidential structure* as a quadruple $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$, where $\mathcal{O}$ is the finite set of possible observations, $m_{\mathcal{O}}$ is an initially basic belief assignment on $2^{\mathcal{O}}$, W is the frame of discernment, and Γ is a multi-valued mapping from $\mathcal{O}$ into W that associates to each element Ob in $\mathcal{O}$ a subset $\Gamma(Ob)$ of W. For any $O \in 2^{\mathsf{Dom}(\Gamma)}$ we call the set $\Gamma(O)$ to be *observable* in W, and if A is observable we denote

$$\Gamma^{-1}(A) = \{Ob \in \mathsf{Dom}(\Gamma) | \cup \Gamma(Ob) = A\}.$$

An observation $Ob \in \mathcal{O}$ is said to be *irrelevant* (resp., *relevant*) if $\Gamma(Ob) = \emptyset$ (resp., $\Gamma(Ob) \neq \emptyset$). Naturally, we do not consider any irrelevant observations in the basic evidential structure, i.e. that we assume as an assumption that every observations in the basic evidential structure is relevant. However, irrelevant observations may occur once the conditioning information from a new piece of evidence becomes available. The set of observations $\mathcal{O}$ is said to be *complete* in the basic evidential structure if

$$\bigcup_{Ob \in \mathcal{O}} \Gamma(Ob) = W,$$

and *mutually exclusive* if $\Gamma(Ob) \cap \Gamma(Ob') = \emptyset$ for any $Ob, Ob' \in \mathcal{O}$ and $Ob \neq Ob'$. Intuitively, the set of observations $\mathcal{O}$ is incomplete when the available observations do not cover completely the situation, and the true state of the world may be in $W \setminus \Gamma(\mathcal{O})$. Consequently, a positive belief mass may be assigned to $\emptyset$, i.e. $m_{\mathcal{O}}(\emptyset) > 0$ that corresponds to the so-called *open world assumption*. Hereafter we accept the *closed world assumption*, namely $m_{\mathcal{O}}(\emptyset) = 0$.

Given a basic evidential structure $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$, as our main concern is on W, the initially basic belief assignment $m_{\mathcal{O}}$ should be propagated to W in a natural way similar to the case of Dempster's approach. For any $A \in 2^W$, the set $\Gamma_*(A) = \{Ob \in \mathcal{O} | \Gamma(Ob) \subseteq A\}$[5] consists of all observations that, according to available evidence, support (imply) the proposition "ϖ is in A", and the set

[5] Note that, by assumption, $\mathsf{Dom}(\Gamma) = \mathcal{O}$.

oarieli@mta.ac.il

$\Gamma^*(A) = \{Ob \in \mathcal{O} | \Gamma(Ob) \cap A \neq \emptyset\}$ consists of all observations in which the proposition is possible. It is clearly that any nonempty subsets of $\Gamma_*(A)$ also support the proposition, whilst any subsets of $\mathcal{O}$ having a nonempty intersection with $\Gamma^*(A)$ cause the proposition possible. Thus we can define the *degree of support* and the *degree of plausibility* for A, denoted by $Sp(A)$ and $Pl(A)$, respectively, as follows

$$Sp(A) = bel_{\mathcal{O}}(\Gamma_*(A)) \tag{8}$$

$$Pl(A) = pl_{\mathcal{O}}(\Gamma^*(A)) \tag{9}$$

where $bel_{\mathcal{O}}$ and $pl_{\mathcal{O}}$ respectively are the belief function and the plausibility function defined on $2^{\mathcal{O}}$ from $m_{\mathcal{O}}$.

Remark 4.1. – When the belief is probabilistic ([28], page 222), a basic evidential structure becomes a Dempster's structure. More especially, the ideal situation where $\mathcal{O}$ is finest, i.e. each observation covers exactly one possible state of the world, induces a probability model.
– If the set of observations $\mathcal{O}$ in the structure $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ is complete and mutually exclusive, the model is reduced to the TBM as shown below.
– If the set of observations $\mathcal{O}$ is complete and mutually exclusive and $bel_{\mathcal{O}}$ is a probability function, then observable subsets in W become measurable events and the PBM without the dynamic component is equivalent to Fagin and Halpern's inner and outer measures model.

Interestingly enough, we have the following theorem.

Theorem 4.1. *Let $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ be a basic evidential structure. Then we have $Sp : 2^W \to [0, 1]$ with $Sp(A) = bel_{\mathcal{O}}(\Gamma_*(A))$ is a belief function.*

Proof. Clearly Sp satisfies B1, i.e. $Sp(\emptyset) = 0$ and $Sp(W) = 1$, so it suffices to show that it satisfies B2. Given subsets $A_1, A_2, \ldots, A_n \in 2^W$, we now show that

$$Sp(\bigcup_{i=1}^{n} A_i) \geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} Sp(\bigcap_{i \in I} A_i).$$

Indeed, by definition we have

$$\Gamma_*(\bigcup_{i=1}^{n} A_i) \supseteq \bigcup_{i=1}^{n} \Gamma_*(A_i)$$

and, for any $\emptyset \neq I \subseteq \{1, \ldots, n\}$,

$$\Gamma_*(\bigcap_{i \in I} A_i) = \bigcap_{i \in I} \Gamma_*(A_i).$$

These follow that

$$\begin{aligned} Sp(\bigcup_{i=1}^{n} A_i) = bel_{\mathcal{O}}(\Gamma_*(\bigcup_{i=1}^{n} A_i)) &\geq bel_{\mathcal{O}}(\bigcup_{i=1}^{n} \Gamma_*(A_i)) \\ &\geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} bel_{\mathcal{O}}(\bigcap_{i \in I} \Gamma_*(A_i)) \\ &= \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} bel_{\mathcal{O}}(\Gamma_*(\bigcap_{i \in I} A_i)) \\ &= \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} Sp(\bigcap_{i \in I} A_i) \end{aligned}$$

This proves the theorem.

oarieli@mta.ac.il

Note that Sp and Pl also form a dual pair, i.e. $Sp(A) = 1 - Pl(\overline{A})$ for any $A \in 2^W$. Thus Pl is a plausibility function.

The duality shows an one-to-one correspondence between these two functions. The plausibility function is just another way of presenting the same information as the support function doing and so could be forgotten.

4.2 Conditioning as Belief Revision

Now assume that some further evidence becomes available and implies that $B \in 2^W$ is surely true. In the PBM, an observation Ob in $\mathcal{O}$ such that $\Gamma(Ob) \cap B = \emptyset$ becomes irrelevant in the light of new evidence. More particularly, the conditioning on B means that the mapping $\Gamma : \mathcal{O} \to 2^W$ has been transformed into the mapping $\Gamma_B : \mathcal{O} \to 2^W$ with $\Gamma_B(Ob) = \Gamma(Ob) \cap B$[6]. As B is surely true in the light of new evidence, Your evidential corpus ($\mathrm{EC}_t^{\mathrm{Y}}$) must be revised according to B. Thus the new evidence should be propagated back to $2^{\mathcal{O}}$ and results in, following Smets' proposal, the mass $m_{\mathcal{O}}(O)$ initially allocated to O is then transferred to $O \cap \Gamma^*(B)$. Clearly, $\Gamma^*(B) = \mathsf{Dom}(\Gamma_B)$. The initially basic belief assignment $m_{\mathcal{O}}$ is transformed into $m_{\mathcal{O}}(\cdot|\Gamma^*(B)) : 2^{\mathcal{O}} \to [0,1]$ with

$$m_{\mathcal{O}}(O|\Gamma^*(B)) = \begin{cases} c \sum\limits_{X \subseteq \overline{\Gamma^*(B)}} m_{\mathcal{O}}(O \cup X) & \text{for} \quad O \subseteq \Gamma^*(B), \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where

$$c = \frac{1}{1 - \sum\limits_{X \subseteq \overline{\Gamma^*(B)}} m_{\mathcal{O}}(X)}.$$

The rule of conditioning is expressed in terms of the belief function $bel_{\mathcal{O}}$ as follows

$$bel_{\mathcal{O}}(O|\Gamma^*(B)) = \frac{bel_{\mathcal{O}}(O \cup \overline{\Gamma^*(B)}) - bel_{\mathcal{O}}(\overline{\Gamma^*(B)})}{1 - bel_{\mathcal{O}}(\overline{\Gamma^*(B)})}$$

On the other hand, the conditioning on B with respect to Sp yields, according to Dempster's rule of conditioning, the following

$$Sp(A|B) = \frac{Sp(A \cup \overline{B}) - Sp(\overline{B})}{1 - Sp(\overline{B})} \tag{11}$$

The following theorem shows that the propagation of conditioning is consistent with the transfer of beliefs. Hence the name PBM.

Theorem 4.2. *Let $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ be a basic evidential structure. Then the rule of conditioning as belief revision above is consistent with the transfer of beliefs.*

[6] This goes back to Dempster [2].

oarieli@mta.ac.il

Proof. Given $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ and a new piece of evidence that implies B is surely true. Then by the rule of conditioning as belief revision we have

$$\begin{aligned} Sp_B(A) &= bel_{\mathcal{O}}(\Gamma_{B*}(A)|\Gamma^*(B)) \\ &= \frac{bel_{\mathcal{O}}(\Gamma_{B*}(A) \cup \overline{\Gamma^*(B)}) - bel_{\mathcal{O}}(\overline{\Gamma^*(B)})}{1 - bel_{\mathcal{O}}(\overline{\Gamma^*(B)})} \end{aligned} \tag{12}$$

On the other hand, it follows by definition that

$$\begin{aligned} Sp(\overline{B}) &= bel_{\mathcal{O}}(\Gamma_*(\overline{B})) \\ &= bel_{\mathcal{O}}(\overline{\Gamma^*(B)}) \end{aligned} \tag{13}$$

Furthermore, it is easy to check the following holds

$$\Gamma_*(A \cup \overline{B}) = \Gamma_{B*}(A) \cup \overline{\Gamma^*(B)}$$

That immediately implies

$$Sp(A \cup \overline{B}) = bel_{\mathcal{O}}(\Gamma_{B*}(A) \cup \overline{\Gamma^*(B)}) \tag{14}$$

The equations (13) and (14) imply that $Sp_B(A) = Sp(A|B)$ (review (11) and (12)). In terms of basic belief assignments, we obtain the following schema

$$\begin{array}{ccc} m_{\mathcal{O}} & \xrightarrow{\text{propagation}} & m(Sp) \xrightarrow{\text{transfer}} m_B(Sp_B) \\ \downarrow \text{transfer} & & \| \\ m_{\mathcal{O}}(\cdot|\Gamma^*(B)) & \xrightarrow{\qquad\text{propagation}\qquad} & m(\cdot|B)(Sp(\cdot|B)) \end{array}$$

This concludes the proof.

Remark 4.2. The PBM is reduced to the TBM when once the set of observations $\mathcal{O}$ in a basic evidential structure $(\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ is complete and mutually exclusive. Indeed, since $\mathcal{O}$ is complete and mutually exclusive, i.e. $\{\Gamma(Ob)|Ob \in \mathcal{O}\}$ forms a partition of W, hence the set of observable subsets in W forms a Boolean algebras that is isomorphic to $2^{\mathcal{O}}$. Thus, it is legitimate to define $m : 2^W \longrightarrow [0, 1]$ as follows

$$m(A) = \begin{cases} m_{\mathcal{O}}(\Gamma^{-1}(A)) & \text{if } A \text{ is observable,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that if A is observable then $\Gamma^{-1}(A) = \Gamma^*(A) = \Gamma_*(A)$. Consequently, $(W, \mathcal{R}, Sp)$ is a credibility space in the sense of Smets and Kennes [28], where $\mathcal{R}$ is the Boolean algebra of the observable subsets of W generated by $\Gamma(\mathcal{O})$.

Remark 4.3. In the case where focal elements of $m_{\mathcal{O}}$ are exactly singletons, i.e. that $bel_{\mathcal{O}}$ is a probability function, a basic evidential structure becomes a Dempster's structure. Then Dempster considered that the conditioning on $B \subseteq W$ means the transformation of Γ into Γ_B, and also postulated that the knowledge of the conditioning event B does not modify $bel_{\mathcal{O}}$. This opened to criticism [29]. Surprisingly, whilst Dempster defined the lower probability of an event A in W as the conditional probability of $\Gamma_*(A)$ given the set of only relevant observations $\mathsf{Dom}(\Gamma)$ in $\mathcal{O}$ (review (3)), he did not take the idea into account once the conditioning information becomes available.

oarieli@mta.ac.il

4.3 Refinements and the Three Prisoners Problem

Let us consider two basic evidential structures $BE_1 = (\mathcal{O}_1, m_{\mathcal{O}_1}, W, \Gamma_1)$ and $BE = (\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$ on the same frame of discernment W. We call BE_1 is a *refinement* of BE if there is a surjection $f : \mathcal{O}_1 \to \mathcal{O}$ such that $\Gamma = \Gamma_1 \circ f^{-1}$ and $bel_{\mathcal{O}}(O) = bel_{\mathcal{O}_1}(f^{-1}(O))$, for any $O \in 2^{\mathcal{O}}$. An illustrated example is depicted as below.

We would like to close this section by analyzing the well-known *three prisoners problem*, that is one of the most quoted examples concerning the applicability of Dempster's rule of conditioning, e.g. [18,7]. The problem is stated as follows[7].

> Let a, b and c be three prisoners. Two of the prisoners are chosen by the warden to be executed but a does not know which. He therefore says to the jailer: "Since either b or c is certainly going to be executed, you will give me no information about my own chances if you give me the name of one man, either b or c, who is going to be executed." Accepting this argument, the jailer truthfully replies: "b will be executed." Thereupon a feels happier because before the jailer replied, his own chance of execution was two-thirds, but afterwards there are only two people, himself and c, who could be the one not executed, and so his chance of execution is one-half.

Is the prisoner a justified in believing that his chance of escaping has improved?

Before analyzing the problem in terms of a basic evidential structure. We note that, as discussed in [6], in order for a to believe that his own chance of execution was two-thirds before the jailer replied, he seems to be implicitly assuming that the one to get pardoned is chosen at random from among a, b and c. This assumption means that each prisoner would be randomly selected with probability $\frac{1}{3}$ to be pardoned. Further, following [6] we model a possible state by a pair (x, y), where $x, y \in \{a, b, c\}$, that represents a state where x is pardoned and the jailer replies that y will be executed to a's question. Since the jailer answers truthfully and will never tell a directly that a will be executed, we have the set of possible states is $W = \{(a, b), (a, c), (b, c), (c, b)\}$.

We now construct a basic evidential structure for a before getting the answer from the jailer as $BE = (\mathcal{O}, m_{\mathcal{O}}, W, \Gamma)$, where $\mathcal{O} = \{Ob_a, Ob_b, Ob_c\}$ with Ob_x corresponds to "x is pardoned", $m_{\mathcal{O}}(Ob_x) = \frac{1}{3}$, for every $x \in \{a, b, c\}$, and $\Gamma(Ob_a) = \{(a, b), (a, c)\}$, $\Gamma(Ob_b) = \{(b, c)\}$, $\Gamma(Ob_c) = \{(c, b)\}$. Let us denote *says-b* the event $\{(a, b), (c, b)\}$ corresponding to the jailer's answer. Then two situations could be arisen when the jailer gave the answer to a's question [26].

Context 1. a has learnt that the jailer's answer is surely true (e.g., the jailer saw the result of the selection from the judge), i.e. that *says-b* is surely true. Then a should revise his belief by conditioning on *says-b* from BE, that results in

$$Sp_{says\text{-}b}(\{(a, b), (a, c)\}) = \frac{1}{2}.$$

Hence he feels happier realistically.

[7] This description of the story is taken from [6] and our discussion is based on that of Fagin and Halpern [6] and Smets [26]

oarieli@mta.ac.il

Context 2. a has learnt that the jailer chooses at random between saying b and c if a is pardoned. This is because the jailer would like to satisfy the prisoner a while making sure that the answer does not change a's belief about his chance of saving. Then a has been just updated a piece of uncertain information that "the probability that jailer chooses at saying b will be executed is $\frac{1}{2}$". This uncertain information helps a just refining his basic evidential structure, say $BE' = (\mathcal{O}', m_{\mathcal{O}'}, W, \Gamma')$, that is a refinement of BE, where $\mathcal{O}' = \{Ob_{ab}, Ob_{ac}, Ob_b, Ob_c\}$ with Ob_{ax} corresponds to "a is pardoned and the jailer says x", $m_{\mathcal{O}'}(Ob_{ax}) = \frac{1}{6}$, $m_{\mathcal{O}'}(Ob_x) = \frac{1}{3}$ for $x \in \{b, c\}$, and $\Gamma'(Ob_{ab}) = \{(a, b)\}$, $\Gamma'(Ob_{ac}) = \{(a, c)\}$, $\Gamma'(Ob_b) = \{(b, c)\}$, $\Gamma'(Ob_c) = \{(c, b)\}$. This yields a probability model, and then one gets

$$Sp_{says\text{-}b}(\{(a, b), (a, c)\}) = \frac{Sp(\{(a, b)\})}{Sp(\{(a, b), (c, b)\})} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}.$$

5 Conclusions

In this paper we have proposed a new approach to belief modeling based on Smets' view of the origin of beliefs and the notion of a multi-valued mapping. Interestingly enough, the model also induces a belief function that quantifies our degrees of support in subsets of the frame of discernment given a basic evidential structure. Furthermore, it has been shown that the propagation of conditioning in the model is consistent with the transfer of beliefs.

As we have mentioned in Remarks 4–5, the approach proposed in this paper has also provided a generalization of a number of existing models. This may allow us to understand their commonalities and differences, and to facilitate the formal comparison of these models. We do hope that this will also support a better understanding of existing models of beliefs and serve as a bridge of the gap between well-known approaches. More details on the model as well as the problem on the combination of evidence in the model will be presented in a forthcoming paper.

References

1. DeGroot, M. H., *Optimal Statistical Decisions* (McGraw-Hill, New York, 1970).
2. Dempster, A. P., Upper and lower probabilities induced by a multi-valued mapping, *Annals of Mathematics and Statistics* **38** (1967) 325–339.
3. Dempster, A. P., A generalization of Bayesian inference, *Journal of the Royal Statistical Society, Series B* **30** (1968) 205–247.
4. Elouedi, Z., K. Mellouli, and P. Smets, Belief decision trees: theoretical foundations, *International Journal of Approximate Reasoning* **28** (2001) 91–124.
5. Fagin, R., and J. Y. Halpern, Uncertainty, belief, and probability, *Computational Intelligence* **7** (1991) 160–173.
6. Fagin, R., and J. Y. Halpern, A new approach to updating beliefs, in P. P. Bonissone et al. (Eds.), *Uncertainty in Artificial Intelligence* 6, 1991, pp. 347–374.
7. Gebhardt, J., and R. Kruse, The context model: An integrating view of vagueness and uncertainty, *International Journal of Approximate Reasoning* **9** (1993) 283–314.

oarieli@mta.ac.il

8. Halmos, P., *Measure Theory* (Van Nostrand, 1950).
9. Halpern, J. Y., and R. Fagin, Two views of belief: belief as generalized probability and belief as evidence, *Artificial Intelligence* **54** (1992) 275–317.
10. Halpern, J. Y., and M. O. Rabin, A logic to reason about likelihood, *Artificial Intelligence* **32** (3) (1987) 379–405.
11. Huynh, V. N., M. Ryoke, Y. Nakamori, and T. B. Ho, Fuzziness and uncertainty within the framework of context model, in: T. Bilgic et al. (Eds.), *Fuzzy Sets and Systems – FSS 2003*, LNAI 2715 (Springer-Verlag, Berlin Heidelberg, 2003) 219–228.
12. Kohlas, J., Support and plausibility functions induced by filter-valued mappings, *International Journal of General Systems* **21** (4) (1993) 343–363.
13. Kohlas, J., and P. A. Monney, Representation of evidence by hints, in R. R. Yager et al. (Eds.), *Advances in the Dempster-Shafer Theory of Evidence* (John Wiley, New York, 1994) 473–492.
14. Kohlas, J., and P. A. Monney, *A Mathematical Theory of Hints: An Approach to Dempster-Shafer Theory of Evidence*, Lecture Notes in Economics and Mathematical Systems **425** (Springer-Verlag, Berlin-Heidelberg, 1995).
15. Murai, T., M. Miyakoshi, and M. Shimbo, Soundness and completeness theorems between the Dempster-Shafer theory and logic of belief, in: *Proceedings of the Third IEEE Conference on Fuzzy Systems*, Orlando, Florida, June 1994, 855–858.
16. Nilsson, N., Probabilistic logic, *Artificial Intelligence* **28** (1986) 71–87.
17. Nguyen, H. T., On random sets and belief functions, *Journal of Mathematical Analysis and Applications* **65** (1978) 531–542.
18. Pearl, J., Reasoning with belief functions: an analysis of compatibility, *International Journal of Approximate Reasoning* **4** (1990) 363–390.
19. Rasiowa, H., *An Algebraic Approach to Non-classical Logic* (North-Holland, Amsterdam-New York, 1974).
20. Resconi, G., G. Klir, and U. St. Clair, Hierarchically uncertainty metatheory based upon modal logic, *International Journal of General Systems* **21** (1992) 23–50.
21. Ruspini, E. H., *The logical foundations of evidential reasoning*, Technical Report 408, AI Center, SRI International, Menlo Park, California, 1986.
22. Shafer, G., *A Mathematical Theory of Evidence* (Princeton University Press, Princeton, 1976).
23. Shafer, G., Allocations of probability, *Annals of Probability* **7** (5) (1979) 827–839.
24. Smets, P., Belief functions, in: P. Smets et al. (Eds.) *Non Standard Logics for Automated Reasoning* (Academic Press, London, 1988) 253–286.
25. Smets, P., The combination of evidence in the transferable belief model, *IEEE Transactions on Patter Analysis and Machine Intelligence* **12** (5) (1990) 447–458.
26. Smets, P., Resolving misunderstandings about belief functions: a response to the many criticisms raised by J. Pearl, *International Journal of Approximate Reasoning* **6** (1992) 321–344.
27. Smets, P., Quantifying beliefs by belief functions: An axiomatic justification, in *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (San Mateo, CA, Morgan Kaufmann, 1993) 598–603.
28. Smets, P., and R. Kennes, The transferable belief model, *Artificial Intelligence* **66** (1994) 191–234.
29. Smets, P., What is Dempster–Shafer's model, in R. R. Yager et al. (Eds.), *Advances in the Dempster-Shafer Theory of Evidence* (John Wiley, New York, 1994) 5–34.
30. Smets, P., The normative representation of quantified beliefs by belief functions, *Artificial Intelligence* **92** (1997) 229–242.

oarieli@mta.ac.il

31. Smets, P., and R. Kruse, The transferable belief model for belief representation, in A. Motro and P. Smets (Eds.), *Uncertainty Management in Information Systems: From Needs to Solutions* (Kluwer, Boston, 1997) 343–368.
32. Zadeh, L. A., Fuzzy logics and approximate reasoning, *Synthese* **30** (1975) 407–428.

oarieli@mta.ac.il

Computer-Oriented Calculi of Sequent Trees

Alexander Lyaletski

Faculty of Cybernetics, Kyiv National Taras Shevchenko University,
2, Glushkov avenue, building 6, 03022 Kyiv, Ukraine
tel.: (38 044) 259-0530, fax: (38 044) 259-0439
lav@unicyb.kiev.ua

Abstract. The problem of construction of a computer-oriented technique for inference search based on a certain sequent formalism for first-order classical logic with equality is solved. For this, special calculi of so-called sequent trees are constructed. The following features are inherent to the tree calculi: (i) preliminary skolemization is used for increasing their proof search efficiency with the help of a technique for finding the most general simultaneous unifier, (ii) every calculus is completely induced by a correspondent sequent calculus, (iii) any transformation of a sequent tree is defined by an appropriate (propositional) rule, and what's more, (iv) certain kinds of the paramodulation rule are added to the sequent tree calculi. For all the calculi, some results about their soundness and completeness are given. Note that an approach under consideration can give a possibility to incorporate the proposed paramodulution technique into, for example, different modifications of the model elimination method, of goal-oriented sequent calculi, and of the tableaux method.

1 Introduction

At present, elaboration of inference search methods, which foresee a participation of man in inference searching, is given more consideration. However, when for this purpose one tries to use well-known methods relying upon results of Skolem [1] and Herbrand [2] and having a sufficiently high efficiency (such as resolution-type methods, the inverse method, connection graph methods, etc.), a number of difficulties arise. These difficulties are caused by the fact that both the specificity of the methods consisting in the "destruction" of an assertion to be proven by transforming it into clauses, collections, connection graphs, an so on, and ways of organizing a proof search process impede the implementation of tools for the construction of such a "natural" deduction, which could be achieved by using Gentzen calculi [3] (for example, Kanger's calculus [4]).

At the same time, usual Gentzen-type sequent calculi significantly yield proof search efficiency, for example, to resolution-type methods. In general, this is connected with additional search efforts caused, on the one hand, by the possibility of different orders of quantifier rule applications, which arises due to absence of preliminary skolemization, and, on the other hand, by uncontrolled selection

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 213–230, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

of a propositional rule for its application. This situation is made worse when necessity for efficient equality manipulation appears. That is why last time the state of automated reasoning is characterized by the fact that the great attention attracts construction of proof search methods combining the best features of first-order sequent formalism and logical combinatory technique (for example, sophisticated types of resolution and paramodulation techniques). These words are confirmed by the great number of appropriate publications. (For this, it is enough to turn to the Handbook of Automated Reasoning (ed. by A.Robinson and A. Voronkov) containing papers, such as [5],[6],[7],[8], and so on.)

In this paper an attempt is made to investigate some properties of sequent inferences in the form of trees considering them as well-formed expressions of special (sequent) tree calculi constructed for the effective establishment of the deducibility of sequents in first-order classical logic (with or without equality).

The sequent trees calculi considered in this paper have the following features: skolemization is used for eliminating non-effective quantifier manipulations, every tree calculus is completely induced by a correspondent sequent calculus in the usual form, any transformation of a sequent tree is goal-driven, and what's more, certain kinds of the paramodulation rule are incorporated in the sequent tree calculi. Note that we concentrate our attention only on the most general (fundamental) ways of adding the paramodulation to the sequent tree calculi, disregarding methods of building-in different well-studied sophisticated modifications of the paramodulation rules, which is explained by fact that any paramodolution modification requires separate thorough study.

For all the calculi under consideration, some results about their soundness and completeness are given.

2 Preliminaries

The sequent form of first-order classical logic with equality is considered. Its language includes the universal and existential quantifiers ($\forall x$ and $\exists x$), the propositional connectives of implication ($\supset$),disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$), and the equality symbol ($=$). Below, atomic formulas are denoted by A, literals are denoted by L or M, formulas are denoted by English capital letters. Sequences of formulas are denoted by Greek capital letters. All letters can be subscripted.

2.1 General Notions

Notions of terms, atomic formula, formulas, and literals are considered to be known. A formula being the result of renaming of variables in some formula is called its *variant*.

The empty formula is denoted by $\sharp$.

The expression $F^{\neg}$ denotes the result of carrying the negation into a formula F: $(\forall x P)^{\neg}$ is $\exists x P^{\neg}$, $(\exists x P)^{\neg}$ is $\forall x P^{\neg}$, $(P \supset Q)^{\neg}$ is $P \wedge Q^{\neg}$, $(P \vee Q)^{\neg}$ is $P^{\neg} \wedge Q^{\neg}$, $(P \wedge Q)^{\neg}$ is $P^{\neg} \vee Q^{\neg}$, $(\neg P)^{\neg}$ is P, and $A^{\neg}$ is $\neg A$.

oarieli@mta.ac.il

We define *positive* ($P\lfloor F^+\rfloor$) *and negative* ($P\lfloor F^-\rfloor$) *occurrences* of a by the following way:

I. $F\lfloor F^+\rfloor$ always holds.

II. If F occurs in P then:

$P\lfloor F^+\rfloor$	implies	$(\neg P)\lfloor F^-\rfloor$	$P\lfloor F^-\rfloor$	implies	$(\neg P)\lfloor F^+\rfloor$
$P\lfloor F^+\rfloor$	implies	$(P \wedge Q)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(P \wedge Q)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(Q \wedge P)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(Q \wedge P)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(P \vee Q)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(P \vee Q)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(Q \vee P)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(Q \vee P)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(P \supset Q)\lfloor F^-\rfloor$	$P\lfloor F^-\rfloor$	implies	$(P \supset Q)\lfloor F^+\rfloor$
$P\lfloor F^+\rfloor$	implies	$(Q \supset P)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(Q \supset P)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(\forall xP)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(\forall xP)\lfloor F^-\rfloor$
$P\lfloor F^+\rfloor$	implies	$(\exists xP)\lfloor F^+\rfloor$	$P\lfloor F^-\rfloor$	implies	$(\exists xP)\lfloor F^-\rfloor$

Obviously, we can assume that any formula cannot contain two different quantifiers having a common variable.

If $P\lfloor(\forall xF)^+\rfloor$ ($P\lfloor(\exists xF)^-\rfloor$) holds for some formulas F and P, then the quantifier $\forall x$ ($\exists x$) is said to be *positive* in the formula P.

If $P\lfloor(\exists xF)^+\rfloor$ ($P\lfloor(\forall xF)^-\rfloor$) holds for some formulas F and P, then the quantifier $\exists x$ ($\forall x$) is said to be *negative* in the formula P.

An *equation* is a pair of terms s, t written as $s \approx t$.

We use the expression $L(t_1, \ldots, t_n)$ to denote that t_1,...,t_n is a list of all the terms (possibly, with repetitions) occupying argument places in a literal L in the order of their occurrences in L.

If L is a literal, then $\sim L$ denotes its complement.

Assume L is a literal of a form $R(t_1, \ldots, t_n)$ ($\neg R(t_1, \ldots, t_n)$) and M is a literal of a form $R(s_1, \ldots, s_n)$ ($\neg R(s_1, \ldots, s_n)$), where R is a predicate symbol and $t_1, \ldots, t_n, s_1, \ldots, s_n$ are terms. Then $\Sigma(L, M)$ denotes the set of equations $\{t_1 \approx s_1, \ldots, t_n \approx s_n\}$. In this case L and M are said to be *equal modulo* $\Sigma(L, M)$ ($L \approx M$ *modulo* $\Sigma(L, M)$).

We understand sequents in the usual sense. Formulas in antecedent of any sequent are called *premises*, and formulas in its succedent are called *goals* of the sequent. Sequences of premises and goals are thought as sets. So, an order of writing premises and goals is immaterial. Also clearly, that we can think that all the formulas from premises and goals of the same sequent *pairwise have no common variables*.

Let S denote a sequent $\Gamma \to \Delta$ where Γ and Δ are sequences of formulas. If a quantifier $\forall x$ ($\exists x$) is *positive (negative)* in some formula from Δ, then the quantifier is said to be positive (negative) in S. If a quantifier $\forall x$ ($\exists x$) is positive (negative) in some formula from Γ, then the quantifier is said to be *negative (positive)* in S.

Without loss of generality, we can restrict ourselves by considering only *one-goal sequents*, that is sequents with exactly one goal in their succedents. The notion of usual sequents is extended to sequents of the form $\Gamma \to \sharp$, where Γ is a sequence of formulas, and $\sharp$ is the empty formula. These sequents are called *terminal sequents*.

oarieli@mta.ac.il

We consider the reader familiar with the notions of substitution, unifier and most general simultaneous unifier, which are treated like in [9]. Moreover, if E denotes any expression, and σ is a substitution, then the result of application σ to E is understood in the sense of [9] and is denoted by $E * \sigma$. For any set Ex of expressions, $Ex * \Sigma$ denotes the result of application σ to every expression from Ex.

2.2 Herbrand Theorem for One-Goal Sequents

From now on, Herbrand theorem given below in a form extracted from [10] uses.

It is known [10] that by means of special kind of skolemization, the establishment of deducibility of any sequent can be reduced to the establishment of deducibility of a sequent with eliminated positive quantifiers and with free variables as constants. That is why the examination of sequents with only negative quantifiers restricts us further. In this connection, we can eliminate all the quantifiers from sequents. Therefore, we can assume that any sequent consists of quantifier-free formulas, which pairwise have no common variable. We will keep to this restriction over the whole paper.

As usual, the Herbrand universe $H(S)$ for a sequent S is defined as the minimal set containing the following terms: (i) $H(S)$ contains every constant from S (if there are no constants in S, then a special symbol, for example, c_0, belongs to $H(S)$), and (ii) for every functional k-arity symbol f occurred in S and any terms t_1, ..., $t_k \in H(S)$, $H(S)$ contains the term $f(t_1, ..., t_k)$.

As a corollary of some results presented in [10], we obtain the following form of Herbrand theorem.

Herbrand theorem (for one-goal sequents). Let S be a sequent of the form $\Gamma \to G$, where Γ is a sequence of formulas, and G is a formula. The sequent S is deducible in the Gentzen calculus LK [3] if and only if there exist the substitution σ and formulas $P_1, ..., P_n$, such that for every i $(1 \leq i \leq n)$ P_i is a variant of some formula from Γ or a variant of $\sim G$, σ substitutes terms from $H(S)$ for all the variables of S, and the sequent $P_1 * \sigma$,...,$P_n * \sigma \to G * \sigma$ (not containing variables) is deducible in LK.

3 Scheme of Construction of Calculi of Sequent Trees

Well-formed expressions of calculi of sequent trees are trees with nodes labeled by sequents. We identify a node with its label and suppose that any tree grows "from top to bottom". Trees with leaves labeled only by terminal sequents are called *terminal trees*. If S is a sequent, then an *initial tree induced by* S is a tree consisting only of a root labeled by S. We suppose that any initial tree cannot be induced by any terminal sequent.

Every sequent calculus under consideration generates an appropriate calculus of sequent trees. Below we restrict ourselves to sequent calculi containing only one-premise inference rules, when the last are read from "top to bottom".

oarieli@mta.ac.il

A set $Eq(Tr)$ of equations is connected with every (inferred) sequent tree Tr. We suppose $Eq(Tr)$ is equal to $\emptyset$ for every initial tree Tr. For an inferred tree Tr different from an initial tree, $Eq(Tr)$ is determined by an appropriate inference rule of a sequent calculus under consideration.

A terminal tree Tr is called a *proof tree* if and only if there exists the most general simultaneous unifier of $Eq(Tr)$.

Now we have all the necessary to give a general scheme of the construction of sequent tree calculi. Let SC denote any sequent calculus. Then a calculus of sequent trees corresponding to SC is defined as follows.

I. *Axioms* of the calculus of sequent trees are certain initial trees.

II. Inference rules:

II.1. *Inference rules induced by sequent calculus.* Let Lf be a leaf of a sequent tree Tr. Let a rule R of SC can be applied to Lf with generating consequences $Lf_1, ..., Lf_m$ $(m > 0)$. If Tr' is obtained from Tr by means of adding of m successors $Lf_1, ..., Lf_m$ to Lf, then Tr' is said to be inferred from Tr by R. A set $Eq(Tr')$ is determined as $Eq(Tr) \cup \Sigma(L, M)$, where $\Sigma(L, M)$ is determined by R of SC.

II.2. *Rule of Contrary Closing (CC).* Let Tr be a sequent tree and Br be branch of Tr with a leaf Lf labeled by a sequent $\Gamma \to L$, where Γ is a sequence of formulas, and L is a literal. Let Br contain a sequent $\Gamma' \to M$, where Γ' is a sequence of formulas, and M is a literal such that $\sim L \approx M$ *modulo* $\Sigma(\sim L, M)$. If Tr' is obtained from Tr by means of adding one successor labeled by $\Gamma \to \sharp$ to Lf, then Tr' is said to be inferred from Tr by CC. A set $Eq(Tr')$ is determined as $Eq(Tr) \cup \Sigma(\sim L, M)$.

Remark 1. Draw your attention to the fact that CC is a rule of every sequent trees calculus under consideration.

A sequence of trees $Tr_1, ..., Tr_n$ is said to be *an inference* in a calculus of sequent trees if and only if Tr_1 is an initial tree, and for $i > 0$ Tr_{i+1} is inferred from Tr_i by some inference rule.

A *sequent* S is considered to *be inferred w.r.t. a calculus of sequent trees* if and only if for some inference $Tr_1, ..., Tr_n$ in the calculus, the following conditions are satisfied: Tr_1 is an initial tree induced by a certain axiom constructed from S, Tr_n is a terminal tree, and there exists the most general simultaneous unifier σ for pairs from a set $Eq(Tr_1) \cup ... \cup Eq(Tr_n)$. If σ is the empty substitution, then $Tr_1, ..., Tr_n$ is called a *propositional inference*.

Giving appropriate sequent-type calculi, we now can introduce calculi of sequent trees interesting us.

oarieli@mta.ac.il

4 Sequent Trees Calculi for Logic without Equality

This section is devoted to first-order classical logic without equality.

4.1 Calculus ST_1

Axioms of ST_1. Let S be a one-goal sequent $\Gamma \to F$, where Γ is a sequent of formulas, and F is a formula. Then an initial tree induced by a sequent $\Gamma, F^{\neg} \to F'$ is called an *axiom w.r.t. S for* ST_1, where F' is obtained from F by renaming all its variables by new variables and besides, F contains at least one variable.

Thus, in order to define ST_1, it remains only to give a sequent calculus interesting us (cf. [11]). This sequent calculus is completely determined by its inference rules below.

Goal-splitting Rules. These rules apply only to the goals of sequents.

$(\to\supset_1)$-rule:
$(\to\supset_2)$-rule:

$$\frac{\Gamma \to F \supset F_1}{\Gamma, F \to \ F_1} \qquad \frac{\Gamma \to F \supset F_1}{\Gamma, {F_1}^{\neg} \to F^{\neg}}$$

$(\to \vee_1)$-rule:

$$\frac{\Gamma \to F \vee F_1}{\Gamma, F^{\neg} \to F_1}$$

$(\to \vee_2)$-rule:

$$\frac{\Gamma \to F \vee F_1}{\Gamma, {F_1}^{\neg} \to F}$$

$(\to \wedge)$-rule:

$$\frac{\Gamma \to F \wedge F_1}{\Gamma \to F \quad \Gamma \to F_1}$$

$(\to \neg)$-rule:

$$\frac{\Gamma \to \neg F}{\Gamma \to F^{\neg}}$$

Premise Duplication Rule. This rule applies only if a formula F contains at least one variable.

$$\frac{\Gamma_1, F\lfloor M^+ \rfloor, \Gamma_2 \to L}{\Gamma_1, F', F\lfloor M^+ \rfloor, \Gamma_2 \to L}$$

where $L \approx M$ *modulo* $\Sigma(L, M)$, and F' is obtained from F by renaming all its variables by new variables.

Auxiliary Goal Rules. Pay your attention to the fact that an order of applications of auxiliary goal rules is "determined" by a literal-goal L from a succedent of a sequent under consideration. This denotes that first of all we fix a positive or negative occurrence of such a literal M in some premise of the sequent that

oarieli@mta.ac.il

$L \approx M$ *modulo* $\Sigma(L, M)$ if L is an atomic formula or the first sign of M is $\neg$, and $\sim L \approx M$ *modulo* $\Sigma(\sim L, M)$ in the opposite case; after that we apply auxiliary goal rules until the rules are applicable w.r.t. this occurrence of M.

$(\supset_1\rightarrow)$-rule:

$$\frac{\Gamma_1, F \supset F_1\lfloor M^+\rfloor, \Gamma_2 \rightarrow L}{\Gamma_1, F_1\lfloor M^+\rfloor, \Gamma_2 \rightarrow L \quad \Gamma_1, \Gamma_2 \rightarrow F}$$

$(\supset_2\rightarrow)$-rule:

$$\frac{\Gamma_1, F\lfloor M^-\rfloor \supset F_1, \Gamma_2 \rightarrow L}{\Gamma_1, F^\neg\lfloor M^+\rfloor, \Gamma_2 \rightarrow L \quad \Gamma_1, \Gamma_2 \rightarrow F_1{}^\neg}$$

$(\vee_1 \rightarrow)$-rule:

$$\frac{\Gamma_1, F \vee F_1\lfloor M^+\rfloor, \Gamma_2 \rightarrow L}{\Gamma_1, F_1\lfloor M^+\rfloor, \Gamma_2 \rightarrow L \quad \Gamma_1, \Gamma_2 \rightarrow F^\neg}$$

$(\vee_2 \rightarrow)$-rule:

$$\frac{\Gamma_1, F\lfloor M^+\rfloor \vee F_1, \Gamma_2 \rightarrow L}{\Gamma_1, F\lfloor M^+\rfloor, \Gamma_2 \rightarrow L \quad \Gamma_1, \Gamma_2 \rightarrow F_1{}^\neg}$$

$(\wedge_1 \rightarrow)$-rule:

$$\frac{\Gamma_1, F\lfloor M^+\rfloor \wedge F_1, \Gamma_2 \rightarrow L}{\Gamma_1, F\lfloor M^+\rfloor, F_1, \Gamma_2 \rightarrow L}$$

$(\wedge_2 \rightarrow)$-rule:

$$\frac{\Gamma_1, F \wedge F_1\lfloor M^+\rfloor, \Gamma_2 \rightarrow L}{\Gamma_1, F_1\lfloor M^+\rfloor, F, \Gamma_2 \rightarrow L}$$

$(\neg \rightarrow)$-rule:

$$\frac{\Gamma_1, \neg(F\lfloor M^-\rfloor), \Gamma_2 \rightarrow L}{\Gamma_1, F^\neg\lfloor M^+\rfloor, \Gamma_2 \rightarrow L}$$

$(\rightarrow \sharp)$-rule:

$$\frac{\Gamma_1, M, \Gamma_2 \rightarrow L}{\Gamma_1, M, \Gamma_2 \rightarrow \sharp}$$

where (in all the above-defined auxiliary goal rules) $L \approx M$ *modulo* $\Sigma(L, M)$ if L is an atomic formula or the first sign of M is $\neg$, and $\sim L \approx M$ *modulo* $\Sigma(\sim L, M)$ in the opposite case.

Remark 2. The auxiliary goal rules imply that one can assume that for an inference $Tr_1, ..., Tr_n$, a set $Eq(Tr_{i+1})$ is equal to $Eq(Tr_i) \cup \Sigma(L, M)$ only if Tr_{i+1} is inferred from Tr_i by the rule $(\rightarrow \sharp)$, and $Eq(Tr_{i+1})$ is equal to $Eq(Tr_i)$ in other cases.

4.2 Calculus ST_2

Axioms of ST_2. Let S be a one-goal sequent $\Gamma \rightarrow F$, where Γ is a sequent of formulas, and F is a formula. Then an initial tree induced by a sequent $\Gamma, F^\neg \rightarrow F'$ is called an *axiom w.r.t. S for* ST_2, where F' is obtained from F by renaming all its variables by new variables.

Thus, in order to define ST_2, it remains only to give a sequent calculus interesting us (cf. [12]). This sequent calculus completely is determined by its inference rules.

Goal-splitting Rules. As in the case of the calculus ST_1, these rules are used for elimination of the principal logical connective from goals. Note that expressions $\lfloor M^+\rfloor$ and $\lfloor M^-\rfloor$ must be taken into account below only for the case of applying an auxiliary goal rule fixing an occurrence of M. (Any fixed occurrence of M determines an order of goal splitting rule applications.)

oarieli@mta.ac.il

$(\rightarrow\supset_1)$-rule:

$$\frac{\Gamma \rightarrow F \supset F_1\lfloor M^-\rfloor}{\Gamma \rightarrow F_1\lfloor M^-\rfloor}$$

$(\rightarrow\supset_2)$-rule:

$$\frac{\Gamma \rightarrow F\lfloor M^+\rfloor \supset F_1}{\Gamma \rightarrow (F\lfloor M^+\rfloor)^\neg}$$

$(\rightarrow \wedge_1)$-rule:

$$\frac{\Gamma \rightarrow F\lfloor M^-\rfloor \wedge F_1}{\Gamma \rightarrow F\lfloor M^-\rfloor \quad \Gamma \rightarrow F_1}$$

$(\rightarrow \wedge_2)$-rule:

$$\frac{\Gamma \rightarrow F \wedge F_1\lfloor M^-\rfloor}{\Gamma \rightarrow F_1\lfloor M^-\rfloor \quad \Gamma \rightarrow F}$$

$(\rightarrow \vee_1)$-rule:

$$\frac{\Gamma \rightarrow F\lfloor M^-\rfloor \vee F_1}{\Gamma \rightarrow F\lfloor M^-\rfloor}$$

$(\rightarrow \vee_2)$-rule:

$$\frac{\Gamma \rightarrow F \vee F_1\lfloor M^-\rfloor}{\Gamma \rightarrow F_1\lfloor M^-\rfloor}$$

$(\rightarrow \neg)$-rule:

$$\frac{\Gamma \rightarrow \neg(F\lfloor M^+\rfloor)}{\Gamma \rightarrow F^\neg\lfloor M^-\rfloor}$$

Auxiliary Goal Rule (*AG*-rule). Unlike ST_1, the calculus ST_2 contains the only auxiliary goal rule:

$$\frac{\Gamma_1, F\lfloor M^+\rfloor, \Gamma_2 \rightarrow L}{\Gamma_1, F, \Gamma_2 \rightarrow (F'\lfloor M^+\rfloor)^\neg}$$

where (i) $L \approx M$ *modulo* $\Sigma(L, M)$ if L is an atomic formula or the first sign of M is $\neg$, and $\sim L \approx M$ *modulo* $\Sigma(\sim L, M)$ in the opposite case, and (ii) $(F'\lfloor M^+\rfloor)$ is obtained from $F\lfloor M^+\rfloor$ by replacing all its variables by new variables keeping a one-one correspondence between old and new variables.

Remark 3. Attract your attention to the fact that unlike ST_1, the calculus ST_2 does not contain an analog of the rule $(\rightarrow \sharp)$. Therefore, terminal trees can be inferred in ST_2 only by applying the rule CC.

4.3 Main Results

As usual, we are interested in a question on deducibility "power" of the constructed calculi of sequent trees. For this, let us use the notion of formula image of sequent from [10].

Let $P_1, ..., P_n$, and F be formulas ($n \geq 0$), and S denote a sequent $P_1, ..., P_n \rightarrow F$. Then a *formula image* $\phi(S)$ of S is the formula $(P_1 \wedge ... \wedge P_n) \supset F$, when $n > 0$, and $\phi(S)$ is F, when $n = 0$. If F is the empty formula $\sharp$, $\phi(S)$ is $\sharp$.

Let Tr be a sequent tree, and $Lf_1, ..., Lf_k$ be all its leaves. A *formula image* $\phi(Tr)$ of Tr is a formula $\phi(Lf_1) \wedge ... \wedge \phi(Lf_k)$. We assume that $F \wedge \sharp$ ($\sharp \wedge F$) is F, where F is a formula (possibly, $\sharp$). Hence, $\phi(Tr)$ is $\sharp$ if Tr is a terminal tree.

We use the standard notions of validity of formulas and consistency of a set formulas. Note that because all our reasoning are connected with the establishment of deducibility (validity), we always interpret $\sharp$ as a valid formula.

oarieli@mta.ac.il

Proposition 4.1. *Let usual formulas (possibly, containing both positive and negative quantifiers)* $P'_1, ...,\ P'_n$, *form a satisfiable set of formulas, and* F' *be a usual formula. Let* $P_1, ...,P_n$,F *denote (quantifier-free) results of skolemization of* $P'_1, ...,P'_n$, F', *respectively. The formula* F' *is the logical consequence of the formulas* $P'_1, ...,P'_n$, *if and only if the sequent* $P_1, ...,P_n \to F$ *is inferred w.r.t. the calculus* ST_1 *(w.r.t. the calculus* ST_2*).*

Proof. Note that the above-described method of construction of the calculi ST_1 and ST_2, using the notion of the most general simultaneous unifier, allows us to consider only the case of propositional inferences in calculi ST_1 and ST_2. This is followed from given Herbrand theorem and from the fact that for any unifier λ for some sets $Ex_1, ..., Ex_k$ of expressions, the set $Ex_1 * \lambda$ is equal to $(Ex_1 * \sigma) * \lambda'$,..., $Ex_k * \lambda$ is equal to $(Ex_k * \sigma) * \lambda'$, where σ if the most general simultaneous unifier for $Ex_1, ..., Ex_k$, and λ' is some substitution. Thus, we can assume that P_1,..., $P_n \to F$ do not contain variables at all.

Let ST denote any of both calculi of sequent trees, and let the conditions of Prop. 1 be applicable for ST. Note that for every sequent tree Tr different from a terminal tree there exists a rule of ST applied to Tr. Also note that for any initial tree Tr_1 induced by any S, $\phi(Tr_1)$ is logically equivalent to $\phi(S)$.

Let us assume that there exists a propositional inference $Tr_1, ..., Tr_n$ in ST with the following properties (Tr_1 is considered to be the initial tree generated by P_1,..., $P_n \to F$):

1. For every i $(1 \leq i < n)$ $\phi(Tr_i)$ is the logical consequence of $\phi(Tr_{i+1})$;

2. Let $\pi(Tr_j)$ denote such a numerical characteristic of Tr_j that the following conditions are satisfied: (2.1) $\pi(Tr_n) = 0$, and (2.2) for every i $(1 \leq i < n)$ there exists j $(i < j \leq n)$ such that $\pi(Tr_j) < \pi(Tr_i)$ if $\pi(Tr_i)$ is not equal to 0.

It is easy to prove Prop. 1 for ST now.

Indeed, the properties 1 and 2 ensure soundness of ST, i.e. they make sure that F is logical consequence of $P_1, ..., P_n$, when Tr_n is a terminal tree.

An inverse assertion (i.e. completeness of ST) is easily proved by induction on π on the basis of properties 1, (2.1), and (2.2).

It is easy to verify that properties 1 and 2 hold for both ST_1 and T_2. Therefore, ST_1 and T_2 are sound calculi.

To obtain the completeness of ST_1, it remains to examine the properties 1,2.1 and 2.2 when $\pi(Tr_i)$ denotes a number of binary connectives in all the formulas of the initial sequent $P_1, ..., P_n \to F$.

As to ST_2, its completeness can be obtained by means of a certain transformation of a proof tree of ST_1 to a proof tree of ST_2. Q.E.D.

To clarify differences between ST_1 and ST_2, we restrict ourselves to the consideration of the propositional case of classical logic.

Let us establish the logical consequence of the formula $\neg F \vee G$ from the formula $(\neg G \wedge F) \supset G$, using both ST_1 and ST_2.

At once, we note that only terminal trees are demonstrated because they give perfect pictures about how corresponding inferences of sequent trees can be constructed in ST_1 and ST_2. Also, the terminal trees do not contain names of inference rules applied.

oarieli@mta.ac.il

Example of inference in ST_1. The terminal tree is constructed below for the sequent $(\neg G \wedge F) \supset G \to \neg F \vee G$.

$$\dfrac{\dfrac{\dfrac{(\neg G \wedge F) \supset G, G \wedge \neg F \to \neg F \vee G}{(\neg G \wedge F) \supset G, G \wedge \neg F, F \to G}\langle axiom\rangle}{G, G \wedge \neg F, F \to G}\langle \text{ by } (\to \vee_1)\rangle \quad \dfrac{\dfrac{G \wedge \neg F, F \to \neg G \wedge F}{\dfrac{G \wedge \neg F, F \to \neg G}{G \wedge \neg F, F \to \sharp}\langle \text{by } (\to \wedge)\rangle \quad \dfrac{G \wedge \neg F, F \to F}{G \wedge \neg F \to \sharp}}\langle \text{by } (\to \wedge)\rangle}{G \wedge \neg F, F \to \sharp}\langle \text{ by } (\supset_1 \to)\rangle}{G \wedge \neg F, F \to \sharp}$$

Here, the first and third leaves are constructed in accordance with $(\to \sharp)$-rule and the second leaf in accordance with the rule CC.

Example of inference in ST_2. The terminal tree is constructed below for the sequent $\Gamma \to \neg F \vee G$, where Γ is the sequence $(\neg G \wedge F) \supset G$, $F \wedge \neg G$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \to \neg F \vee G}{\Gamma \to G}\langle axiom\rangle}{\Gamma \to (\neg G \wedge F) \wedge \neg G}\langle \text{by } (\to \vee_2)\rangle}{\Gamma \to \neg G \wedge F}\langle \text{by } (AG) \text{ applied to } (\neg G \wedge F) \supset G\rangle \quad \Gamma \to \neg G}{\Gamma \to \neg G}\langle \text{by } (\to \wedge_2)\rangle \quad \Gamma \to \sharp}{\Gamma \to \sharp}\langle \text{by } (\to \wedge_1)\rangle \quad \dfrac{\dfrac{\Gamma \to F}{\Gamma \to \neg F \vee G}\langle \text{by } (AG) \text{ applied to } F \wedge \neg G\rangle}{\Gamma \to \neg F}\langle \text{by } (\to \wedge_1)\rangle}{\Gamma \to \sharp}$$

Here, all the leaves are constructed in accordance with the rule CC only.

Corollary 4.1. *An arbitrary formula F is valid if and only if a sequent $\to F'$ is inferred w.r.t. the calculus ST_1 (w.r.t. the calculus ST_2), where F' is the (quantifier-free) result of skolemization of F.*

Corollary 4.2. *Let formulas $P_1, ..., P_n, F$, $P'_1, ..., P'_n$, and F' be taken from Prop. 1. The sequent $P_1, ..., P_n \to F$ is inferred in the calculus LK from [3] if and only if the sequent $P'_1, ..., P'_n \to F'$ is inferred w.r.t. the calculus ST_1 (w.r.t. the calculus ST_2).*

4.4 Modifications of ST_1 and ST_2

Let us consider the following modifications ${ST_1}^\sharp$ and ${ST_2}^\sharp$ by incorporating a new inference rule CD into ST_1 and ST_2.

These modifications serve as a basis for the construction of complete extensions of the SLD-resolution having the form of a certain calculus of SLD-trees and for adding the paramodulation rule to these extensions. The last can serve as an example of the introduction of the paramodulation into different modifications of the model elimination method. Also, this technique can be applied usefully for equality handling in different variants of the tableaux method.

oarieli@mta.ac.il

Let us introduce a new rule.

Chain Deleting rule (CD-rule). Let Tr be a sequent tree, and Br be a branch of Tr with a leaf Lf labeled by a sequent $\Gamma \to \sharp$, where Γ is a sequence of formulas. Let E denote a set $\Sigma(L, M)$ relating to $\Gamma \to \sharp$, and Ch denote the maximal part of Br such that Ch contains ("is ended" by) $\Gamma \to \sharp$, and every node of Ch with a label different from $\Gamma \to \sharp$ has only one successor. If Tr' denotes the result of a deletion of Ch from Tr and there exists the most general simultaneous unifier σ of E, then $Tr' * \sigma$ is said to be an *inference tree* produced by CD-rule, where $Tr' * \sigma$ is the result of applying σ to all the formulas in Tr.

Further, we assume that the *CD-rule already is applied* after any application of $(\to \sharp)$-rule or CC-rule. (In this connection, it can be considered as a "part" of these rules. Hence, sequent trees in $ST_1{}^\sharp$ and $ST_2{}^\sharp$ do not contain sequents of the form $\Gamma \to \sharp$.)

Proposition 4.2. *Let $P_1, \ldots, P_n$ form a satisfiable finite set of formulas. A formula G is the logical consequence of formulas $P_1, \ldots, P_n$ if and only if there exists a sequent tree for the sequent $P_1, \ldots, P_n$, $G^\neg \to G$ that does not contain any node (i.e. that is the empty tree), and that is inferred in the calculus $ST_1{}^\sharp$ ($ST_2{}^\sharp$).*

A proof of Prop. 2 uses Prop. 1 and the notion of formula image of sequents.

Note that for the examples from the previous section, the empty tree will be inferred both in $ST_1{}^\sharp$ and in $ST_2{}^\sharp$.

5 Calculi of Literal Trees

The case when we examine only sequents of the form C_1,...,$C_n \to L_1 \wedge ... \wedge L_k$, requires a separate consideration, where for every i $(1 \le i \le n)$ C_i is $M_{i,1} \vee ... \vee M_{i,r_i}$, and $M_{1,1}, ..., M_{n,r_n}$, $L_1, ..., L_k$ are literals.

Remark 4. As usual, an expression of the form $M_1 \vee ... \vee M_r$ is called a clause. Also note that in the case, when no more than one of literals $M_1, ..., M_r$ is an atomic formula, $M_1 \vee ... \vee M_r$ is called a *positive Horn clause*.

Because any first-order formula can be reduced to the conjunctive (disjunctive) normal form by means of logical-equivalence preserving transformations, it is easy to see that the establishment of the deducibility of any sequent is equivalent to the establishment of the deducibility of a sequent of the form C_1,...,$C_n \to L_1 \wedge ... \wedge L_k \vee ... \vee$, where for every i $(1 \le i \le n)$ C_i is $M_{i,1} \vee ... \vee M_{i,r_i}$, any M_{ij} and L_k have no common variables, and M_{ij} and M_{pq} can have common variables only if i coincides with p. That is why we can investigate the deducibility of these *literal sequents* only. In this connection note that if G is $L_1 \wedge \ldots \wedge L_k$, then $G^\neg$ is $\sim L_1 \vee \ldots \vee \sim L_k$.

Taking the above into account, the calculus ST_1 can be transformed into a *literal sequents calculus LT* having the following rules. (Note that the restriction of ST_2 on the case of clauses results in the same calculus LT.)

oarieli@mta.ac.il

Goal Splitting rule. The rule below generalizes $(\rightarrow \wedge)$-rule from ST_1 (ST_2) in the case, when $\wedge$ is considered as a multiple-place operation.

$(\rightarrow \wedge)$-rule:

$$\frac{\Gamma \rightarrow L_1 \wedge \ldots \wedge L_m}{\Gamma \rightarrow L_1, \ldots, \Gamma \rightarrow L_m}$$

Auxiliary Goals rule. This rule is applied when the goal of a sequent is a literal.

$(\vee \rightarrow)$-rule:

$$\frac{\Gamma_1, A_1 \vee \ldots \vee A_n \vee M \vee B_1 \vee \ldots \vee B_r, \Gamma_2 \rightarrow L}{\Gamma' \rightarrow\sim A'_1, \ldots, \Gamma' \rightarrow\sim A'_n, \Gamma' \rightarrow\sim B'_1, \ldots, \Gamma' \rightarrow\sim B'_r}$$

where $A_1, \ldots, \vee A_n, B_1, \ldots, B_r, L$, and M are literals, Γ' is the sequence $\Gamma_1, A_1 \vee \ldots \vee A_n \vee M \vee B_1 \vee \ldots \vee B_r, \Gamma_2$, $A'_1 \vee \ldots \vee A'_n \vee M' \vee B'_1 \vee \ldots \vee B'_r$ are renaming of $A_1 \vee \ldots \vee A_n \vee M \vee B_1 \vee \ldots \vee B_r$ by new variables, and $L \approx M'$ *modulo* $\Sigma(L, M')$,.

The calculus LT has the same $(\rightarrow \sharp)$- rule and CC-rule, as ST_1 has.

As to the *duplication rule*, $(\vee \rightarrow)$-rule contains it as a rule "built-in" in LT.

Also note that the Goal Splitting rule is applied to an initial sequent only if it is necessary: further, Auxiliary Goal rule is applied again and again, generating literals as new goals. That is why LT was called a *calculus of literal sequents*.

Proposition 5.1. *Let clauses $P_1, \ldots, P_n$ form a satisfiable finite set of clauses. A conjunction of literals G is a logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^{\neg} \rightarrow G$ in LT.*

Proof. According to Prop. 1, there exists a proof tree Tr w.r.t. the sequent $P_1, \ldots, P_n \rightarrow G$ in the calculus ST_1. Obviously that Tr consists of sequents inferred by applications of $(\rightarrow \wedge)$-rules, $(\vee \rightarrow)$-rules, $(\rightarrow \sharp)$-rule, and CC-rule only. It is not hard to transform Tr into a proof tree in LT. Q.E.D.

We can introduce a calculus $LT^{\sharp}$ in the same way that was used for $ST_1^{\sharp}$.

Corollary 5.1. *Let clauses $P_1, \ldots, P_n$ form a satisfiable finite set of clauses. A conjunction of literals G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists such an inference tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^{\neg} \rightarrow G$ in the calculus $LT^{\sharp}$ that does not contain any node (i.e. that is the empty tree).*

5.1 Completeness of SLD-Resolution

The peculiarity of the calculus LT is that antecedents of inferred sequents coincide with the antecedent of an initial sequent, say, Γ. This permits to consider Γ as a set of input clauses and to transform any inference tree Tr in LT into a tree $\gamma(Tr)$ having the same nodes as Tr, but labeled only by goals of corresponding sequents. Such a tree $\gamma(Tr)$ is said to be a *goal-tree* corresponding to Tr and we have an easy way to go from LT to SLD-resolution.

The completeness of SLD-resolution is a well-known result in Logic Programming (see, for example, [13,14]). The propositions below contain its modifications in the form of Prop. 3, Corollary 1, and Corollary 2.

oarieli@mta.ac.il

Corollary 5.2. *(Soundness and Completeness of SLD-resolution.) Let positive Horn clauses $P_1, \ldots, P_n$ form a satisfiable finite set of clauses and G be a conjunction of atomic formulas. The goal G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^{\neg} \to G$ in the calculus LT without any CC-rule applications. In particular, the SLD-resolution is sound and complete.*

Proof. It is obvious, because Tr does not contain sequents inferred by CC-rule applications. Q.E.D.

Corollary 5.3. *Let positive Horn clauses $P_1, \ldots, P_n$ form a satisfiable finite set of clauses, and G be a conjunction of atomic formulas. The goal G is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists an inference tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^{\neg} \to G$ in the calculus $LT^{\sharp}$ (without any CC-rule applications) that does not contain any node (i.e. that is the empty tree).*

Remark 5. Prop. 3 and Corollaries 3, 4, and 5 show that the calculi LT and $LT^{\sharp}$ can be considered as methods of extensions of SLD-resolution by means of adding the very simple rule CC in the case of consideration of finite sets of arbitrary clauses. This feature of the calculi LT and $LT^{\sharp}$ becomes important, when we are interesting in a complete and simple superstructure of tools implementing SLD-resolution having the form of SLD-trees. If we are interested in a general conclusion of SLD-resolution when arbitrary first-order formulas are under consideration, different modifications of the calculus ST_1 (ST_2) and $ST_1^{\sharp}$ ($ST_2^{\sharp}$) can be implemented.

6 Paramodulation in Sequent Tree Calculi

Application of logic with equality for solving different tasks implies, as a rule, construction and/or use of the already known methods for equality handling. In this connection, we demonstrate some ways of incorporating paramodulation-type rules into the above-described sequent tree calculi. Demonstration of these ways makes for the literal calculi LT and $LT^{\sharp}$ only. Moreover, in what follows we consider that the most general simultaneous unifier σ of $\Sigma(L, M')$, if it exists, immediately applies to all the goals of a tree Tr inferred in LT by means of some $(\vee\to)$-rule application generating $\Sigma(L, M')$. Note that the result of this application of σ to Tr is denoted by $Tr{*}\sigma$. (The restriction is caused by the consideration of LT and $LT^{\sharp}$ only and has no principal role: the approach suggested here can easily be extended to the case of the calculi ST_1 and ST_2.)

Below we use the notion of E-satisfibility of a set of formulas (in particular, of clauses) in the form of [15]. This notion permits to tell about E-consequence of one formula from another (from a set of formulas).

We remind that an expression of the form $f(x_1, ..., x_k) = f(x_1, ..., x_k)$, where f is a k-arity functional symbol and $x_1, ..., x_k$ are variables, is called functionally reflexive axiom.

oarieli@mta.ac.il

If S is a sequent, then $Rf(S)$ denotes the set (sequence) of functionally reflexive axioms for all the functional symbols from S, having a positive arity.

The specific feature of the paramodulation extensions proposed is that the "directions" of the paramodulation rule applications as separate rules should be taken into account by the same way as in [16] (i.e. w.r.t. sequent trees) and that to construct complete extensions (in general) only three paramodulation-type rules from the four possible may be used (and this number of rules cannot be decreased).

6.1 Paramodulation-Type Rules

Let us define four types of the paramodulation rule.

P_i- and P_o-paramodulation rules. Let Tr be a tree of sequents, and $\Gamma \to M$ be a label of some of the leaves from Tr, and the literal M is different from $\sharp$. Let us suppose that Γ contains such a clause D that the paramodulation rule [15] is applicable from C to M (from $\sim M$ to C), where M and $\sim M$ are considered as unit clauses, and C is a variant of D, which has no common variables with the labels from Tr. Let σ be a *most general unifier* and $L_1 \vee ... \vee L_m$ be a paramodulant of this paramodulation application from C to M (from $\sim M$ to C), where $L_1, ..., L_m$ are literals. Then a tree deduced from Tr (w.r.t. D) by the *P_i-paramodulation rule* (the *P_o-paramodulation rule*) is said to be the result of the following transformation of $Tr * \sigma$: m immediate successors are added to the selected leaf (with the label $\Gamma \to M * \sigma$), and for every i $(1 \leq i \leq m)$ the sequent $\Gamma \to \sim L_i$ is assigned to the ith successor when the successors are examined from left to right.

P_d- and P_u-paramodulation rules. Let Tr be a tree, and w be some of the branches of Tr, whose leaf label is $\Gamma \to L$, where L is different from $\sharp$. Let us suppose that the branch w contains a node labelled by $\Gamma \to M$ such that the paramodulation rule is applicable from $\sim M$ to L (from $\sim L$ to M), where L and M are literals considered as clauses. Let σ be a *most general unifier*, and L' be the paramodulant of this paramodulation application from $\sim M$ to L (from $\sim L$ to M) Then a tree deduced from Tr by the *P_d-paramodulation rule* (by the *P_u-paramodulation rule*) is said to be the result of the following transformation of $Tr * \sigma$: one immediate successor with the label $\Gamma \to L'$ is added to the selected leaf (with the label $\Gamma \to L * \sigma$).

6.2 Sequent Tree Calculi with Paramodulation-Type Rules

Let us construct sequent tree calculi with paramodulation by adding the above-defined rules P_i, P_o, P_d, and P_u in such combinations that lead to two essentially different classes of calculi, which are optimal in the sense of their structure and the minimal number of paramodulation-type rules, necessary for providing the completeness of the calculi suggested:

oarieli@mta.ac.il

(i) *Calculi LT_d and $LT_d^\sharp$* are constructed by adding P_i, P_o, and P_d to LT and $LT^\sharp$ respectively,

(ii) *Calculi LT_u and $LT_u^\sharp$* are constructed by adding P_i, P_o, and P_u to LT and $LT^\sharp$ respectively.

6.3 Main Results for Calculi with Paramodulation-Type Rules

Proposition 6.1. *Let clauses $P_1, \ldots, P_n$ form a E-satisfiable finite set of clauses. Let G be a conjunction of literals, and S be the sequent $P_1, \ldots, P_n$, $G^\neg$, $\rightarrow G$. The conjunction G is the logical E-consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^\neg$, $x = x$, Rf(S) $\rightarrow G$ in LT_d.*

Proof. The soundness of LT_d can be obtained by using so-called clause images of literal sequents. A proof of its completeness can be produced by induction on the number of propositional connectives in an initial sequent in the same way as was applied for proving the completeness of the linear paramodulation in [15]. Q.E.D.

Proposition 6.2. *Let clauses $P_1, \ldots, P_n$ form a E-satisfiable finite set of clauses. Let G be the conjunction of literals, and S be the sequent $P_1, \ldots, P_n$, $G^\neg$, $\rightarrow G$. The conjunction G is the logical E-consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^\neg$, $x = x$, Rf(S) $\rightarrow G$ in LT_u.*

Proof. The soundness of LT_u is obtained in the same way as in the case of LT_d. Its completeness can be achieved by the transformation of a proof tree in LT_d into a proof tree in tree in LT_u. Q.E.D.

Corollary 6.1. *Let clauses $P_1, \ldots, P_n$ form a E-satisfiable finite set of clauses. Let G be a conjunction of literals, and S be the sequent $P_1, \ldots, P_n$, $G^\neg$, $\rightarrow G$. The conjunction G is a logical E-consequence of $P_1, \ldots, P_n$ if and only if there exists the empty tree w.r.t. the sequent $P_1, \ldots, P_n$, $G^\neg$, $x = x$, Rf(S) $\rightarrow G$ in $LT_d^\sharp$, as well as in $LT_u^\sharp$.*

Remark 6. Presence of functionally reflexive axioms in the formulations of the above-given propositions is the necessary condition for the completeness of both classes of calculi. This demonstrates an example given in the next section. In addition note that adding the rules P_i, P_o, P_d, and P_u to LT and $LT^\sharp$ in combinations different from the above-given ones and containing no more than three paramodulation rules results in violation of the completeness of calculi which can be constructed. The following examples confirm this (below a and b be constants, x be a variable, f, g, h, and s be functional symbols, and R is a predicate symbol):

1. The deducibility of the sequent $R(b), R(a) \rightarrow a \neq b$ cannot be established by using the combination $\{P_i, P_d$, and $P_u\}$,

2. The deducibility of the sequent $a = b, R(a) \rightarrow R(b)$ cannot be established by using the combination $\{P_o, P_d$, and $P_u\}$, and

oarieli@mta.ac.il

3. The deducibility of the sequent $x = x, a = f(a) \vee b = g(b),\ h(a) \neq h(f(f(a))) \to s(b) = s(g(g(b)))$, cannot be established by using the combination $\{P_i$ and $P_o\}$.

7 Example of Deducibility for Equality

Let us establish that $a \neq b$ is an E-consequence of Γ, where Γ is the following formulas sequence written "from top to bottom":
$\neg R_1(g_1(x,x)) \vee \neg R_2(g_2(y,y))$ (1),
$R_1(h_1(z,z))$ (2),
$R_2(h_2(u,u))$ (3),
$g_1(f(a),f(b)) = h_1(f(a),f(b))$ (4),
$g_2(f(a),f(b)) = h_2(f(a),f(b))$ (5).
(a and b are constants, x, y, z, u are variables, R_1 and R_2 are predicate symbols, and f, g_1, g_2, h_1 and h_2 are functional symbols. Parenthesis contain numbers of formulas being to the left of the numbers.)

To do this, let us consider a sequent $\Delta \to a \neq b$, where Δ is Γ, $a = b$, $f(v) = f(v)$ (6).
(The equality $f(v) = f(v)$ is the functionally reflexive axiom for f. The functionally reflexive axioms for all the other functional symbols g_1, g_2, h_1, and h_2, as well as the reflexive axiom $x = x$ are omitted.)

For this sequent we have the following proof tree in LT_d:

$\Delta \to a \neq b$ (7)	$\langle initial sequent \rangle$
$\Delta \to g_1(f(a),f(a)) \neq h_1(f(a),f(b))$ (8)	$\langle P_o$ from (7) to (4)$\rangle$
$\Delta \to g_1(f(a),f(a)) \neq h_1(f(a),f(a))$ (9)	$\langle P_d$ from (7) to (8)$\rangle$
$\Delta \to R_1(h_1(f(a),f(a)))$ (10)	$\langle P_o$ from (8) to (2)$\rangle$
$\Delta \to R_2(g_2(y,y))$ (11)	$\langle(\vee \to)$, applied to (1) and (10)$\rangle$
$\Delta \to R_2(g_2(f(v),f(v)))$ (12)	$\langle P_d$ from (7) to (10)$\rangle$
$\Delta \to R_2(g_2(f(a),f(v)))$ (13)	$\langle P_d$ from (7) to (12)$\rangle$
$\Delta \to R_2(g_2(f(a),f(b)))$ (14)	$\langle P_d$ from (7) to (13)$\rangle$
$\Delta \to R_2(h_2(f(a),f(b)))$ (15)	$\langle P_i$ from (4) to (14)$\rangle$
$\Delta \to R_2(h_2(f(a),f(a)))$ (16)	$\langle P_d$ from (7) to (15)$\rangle$
$\Delta \to \sharp$ (17)	$\langle(\to \sharp)$, applied to (3) and (16)$\rangle$

In accordance with the Corollary 6, $a \neq b$ is an E-consequence of Γ.

Remark 7. Without the functionally reflexive axiom $f(v) = f(v)$, it is impossible to construct at least one proof tree w.r.t. $\Delta \to R_2(g_2(y,y))$ (11) in the calculus LT_d (LT_u) even if other functionally reflexive axioms will be present. This guarantees that the above-given example demonstrate necessity of functionally reflexive axioms for the completeness both LT_d and LT_u.

oarieli@mta.ac.il

8 Conclusion

Summarizing the above-given results, we see that transferring to inference search, which is based on sequent trees, has a number of positive features connected with the possibility of the effective exploitation of sequent tree formalism: skolemization, goal-driven control, and all the information being kept in sequent trees both for CC-type propositional rules and for different kinds of the paramodulation rule, which have forms of possible "directions" (w.r.t. trees) of applying the paramodulation. (Unfortunately, the simplest above-described way of incorporating the paramodulation into the sequent tree calculi requires functionally reflexive axioms for preserving the completeness. But it is "payment" for ease of its incorporation. The author hopes that functionally reflexive axioms can be removed from the sequent tree calculi as the result of further developing of these tree calculi with the help of some analog of the lazy paramodulation [17]).

Besides, the sequent tree formalism can serve as a good basis for constructing similar technique and obtaining results for well-known forms of resolution and paramodulation rules in linear formats. An example of this can serve the paper [18] given insight into a new interpretation of the model-elimination method [19, 20] (or, in terminology of [15], into the OL-method) and linear refutation [21], whose "behavior" can sufficiently easily be simulated in the terms of the literal tree calculus LT and $LT^{\sharp}$, and therefore, their paramodulation reconstruction can be made in the spirit of this paper.

As to SLD-resolution, we must note that results obtained here give an easy way to convert literal (sequent) trees to trees, which can be considered as a general conclusion of the usual notion of SLD-trees. This gives a "key" to the construction of complete resolution and paramodulation extensions of SLD-refutation methods intended for their implementation in Prolog-like systems that are used already in different applied fields.

In the opinion of the author of this paper, the above gives some possibilities for incorporating the paramodulation technique proposed here into different modifications both of tableaux methods and of model elimination methods in order to attempt to increase their inference search efficiency by developing different strategies that select "directed" applications of the paramodulation rule. But such investigations require separate considerations. The same concern a possibility of further developing the above-developed paramodulation technique for construction of sequent tree calculi that remain complete without using functionally reflective axioms.

Finally, we underline again that this paper serves as an example of the fact that sequent tree-like structures keep much useful information for deduction, and this information can be used by different ways for optimizing inference search.

Acknowledgements. This research was supported by the project INTAS 2000-447. The author thanks all the participants of this project.

The author would also like to thank referees whose comments and useful remarks made it possible to improve the quality of this paper.

oarieli@mta.ac.il

References

1. Skolem, T.: Logisch-kombinatorische Untersuchungen uber die Erfullbarkeit oder Beweisbarkeit mathematischer Satze. Skriftner utgit ar Videnskapsselskaper i Kristiania, **4** (1920) 4–36.
2. Herbrand, J.: Recherches sur la Theorie de la Demonstration. Travaux de la Societe des Sciences et de Lettres de Varsovie, Class III, Sciences Mathematiques et Physiques, **33** (1930).
3. Gentzen, G.: Untersuchungen uber das Logische Schliessen (I, II). Math. Zeit. **39** (1934) 176–210, 405–443.
4. Kanger, S.: Simplified proof method for elementary logic. Comp. Program. and Form. Sys.: Stud. in Logic. Amsterdam, North-Holland, Publ. Co. (1963) 87–93.
5. Hahnle R.: Tableaux and related methods. Handbook of Automated Reasoning (Ed. by A.Robinson and A.Voronkov), Elsevier Science Pub. (2001) 101–178.
6. Degtyarev A., Voronkov A.: Equality reasoning and sequent-based calculi. Handbook of Automated Reasoning (Ed. by A.Robinson and A.Voronkov), Elsevier Science Pub. (2001) 609–704.
7. Nieuwenhuis R., Rubio A.: Paramodulation-based theorem proving. Handbook of Automated Reasoning (Ed. by A.Robinson and A.Voronkov), Elsevier Science Pub. (2001) 371–444.
8. Letz R., Stenz G.: Model elimination and connection tableau procedures. Handbook of Automated Reasoning (Ed. by A.Robinson and A.Voronkov), Elsevier Science Pub. (2001) 2017–2116.
9. Robinson, J.: A machine-oriented logic based on resolution principle. Journal of the ACM, **12**, No 1 (1965) 23–41.
10. Mints, G.: Herbrand theorem (in Russian). Mathematical Theory of Logical Inference. Nauka, Moscow (1967) 311–350.
11. Degtyarev, A., Lyaletski, A., Morokhovets, M.: Evidence Algorithm and sequent logical inference search. LNAI, **1705** (1999) 44–61.
12. Lyaletski, A., Paskevich, A.: Goal-driven inference search in classical propositional logic. Proc. of the Inter. Workshop STRATEGIES'2001. Siena, Italy (2001).
13. Apt, K.R., van Emden M. H.: Contributions into the theory of logic programming. J. of the ASM, **3**, No 29 (1982) 841–862.
14. Lloyd, J.V.: Foundations of logic programming. Springer, Berlin (1987).
15. Chang C., Lee R.: Symbolic logic and mechanical theorem proving. Academic Press, New York, San Francisco, London (1973).
16. Lyaletski A.V., Yurchishin V.V.: On complete extension of input refutation (in Russian). Kibernetika, No 4 (1990).
17. Snyder W. and Lynch C.: Goal-directed strategies for paramodolation. The 4th International Conference "Rewriting Technique and Applications (RTA)", LNCS, **488**, Springer-Verlag (1991) 150–161.
18. Lyaletski A.: On paramodulation-type extensions of the linear refutation and the model elimination method. Bulletin of the Kiev University, **2**, Kiev State University (1997) 225–235.
19. Loveland D.V.: Mechanical theorem proving by model elimination. J. of the ACM, **15**, (1968) 236–251.
20. Loveland D.W.: Automated theorem proving: a logical basis. North-Holland, New York (1978).
21. Loveland D.V.: A linear format for resolution. Proc. IRIA Symp. Automatic Demonstration. N.Y.: Springer (1970) 147–162.

oarieli@mta.ac.il

On Updates of Logic Programs: A Properties-Based Approach

Mauricio Osorio and Fernando Zacarías

Universidad de las Américas, CENTIA.
Sta. Catarina Mártir, Cholula, Puebla
72820 México
josorio@mail.udlap.mx
http://mailweb.udlap.mx/~josorio
fzflores@siu.buap.mx

Abstract. We have studied the update operator $\oplus$ defined in [4] without tautologies and we have observed that satisfies an interesting property. This property is similar to one postulate proposed by AGM but, in this case for nonmonotonic logic and that we called WIS. Also, we consider other five additional basic properties about update programs and we show that $\oplus$ satisfies them. So, this work continues the analysis about the AGM postulates with respect to operator $\oplus$ under the refinated view that includes knowledge and beliefs that we began in a recent previous paper and that satisfies the WIS property for closed programs under tautologies.

Keywords: Answer set programming; *Nelson* logic; Update programs; Strong negation; AGM postulates; Properties.

1 Introduction

A-Prolog (Stable Logic Programming [5] or Answer Set Programming) is the realization of much theoretical work on Nonmonotonic Reasoning and AI applications of Logic Programming (LP) in the last 15 years. This is an important logic programming paradigm that has now great acceptance in the community. Efficient software to compute answer sets and a large list of applications to model real life problems justify this assertion. The two most well-known systems that compute Answer sets are `dlv`[1] and `smodels`[2]. It has been recently provided a characterization of answer sets by intuitionistic logic as follows: a literal is entailed by a program in the answer set semantics if and only if it belongs to every intuitionistically complete and consistent extension of the program formed by adding only negated literals [10]. The idea of these completions using in general intermediate logics is due to Pearce [8]. This logical approach provides the foundations to define the notion of nonmonotonic inference of any propositional theory (using the standard connectives) in terms of a monotonic logic (namely

[1] `http://www.dbai.tuwien.ac.at/proj/dlv`
[2] `http://saturn.hut.fi/pub/smodels`

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 231–241, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

intuitionistic logic), see [8,10,9]. The proposed interpretation would be the following: Given a theory T, its knowledge is understood as the formulas F such that F is derived in T using intuitionistic logic. This makes sense, since in intuitionistic logic according to Brouwer, F is identified with "I knows F" (or perhaps some reader would prefer to understand the notion of "knowledge" as "justified belief"). An agent whose knowledge base is the theory T believes F if and only if F belongs to every intuitionistically complete and consistent extension of T by adding only negated literals (here "belief" could be better interpreted as "coherent" belief). Take for instance: $\neg a \rightarrow b$. The agent knows $\neg a \rightarrow b$, $\neg b \rightarrow \neg\neg a$ and so on and so forth. The agent does not know however a. Nevertheless, one believes more than one knows, but a cautious agent must have its beliefs consistent to its knowledge. This agent will then assume negated literals to be able to infer more information. Thus, in our example, our agent will believe $\neg a$ and so he/she can conclude b. It also makes sense that a cautious agent will believe $\neg a$ or $\neg\neg a$ rather than to believe a (recall that a is not equivalent to $\neg\neg a$ in intuitionistic logic). This view seems to agree with a point of view by Kowalski, namely that "Logic and LP need to be put into place: Logic within the thinking component of the observation-thought-action cycle of a single agent, and LP within the belief component of thought". As Pearce noticed, if we include strong negation we just have to move to Nelson logics [8]. We select here N, the least constructive (strong negation) extension of intuitionistic logic and because it is the minimum necessary to satisfy our properties. Also, N is the nearest Intuitionistic logic. For this reason we don't need of N2. We say that two theories T_1 and T_2 are equivalent knowledge if they are equivalent in N. We denote this fact by $T_1 \equiv_K T_2$. We say that T_1 and T_2 are equivalent if they have the same answer sets. We denote this fact by $T_1 \equiv T_2$. We will write $P \vdash_K \alpha$ to denote the fact that $P \vdash_N \alpha$. The idea for using K instead of N is due to two reasons: First, to emphasize the reading P "knows" α. Second, because strictly speaking we are translating the connective symbols. Similarly we will write $P_1 \equiv_K P_2$ instead of $P_1 \equiv_N P_2$.

In this paper we address our approach to update nonmonotonic knowledge bases represented as extended logic programs under the answer set semantics, two very good recent reviews with many references are [2,4]. If new knowledge of the world is somehow obtained, and it doesn't have conflicts with the previous knowledge then this new knowledge only expands knowledge. If by the contrary, new knowledge is inconsistent with the previous knowledge, and we want knowledge to be always consistent so that our agents can act in all moment, we should solve this problem somehow. We point out that new information is incorporated into the current knowledge base subject to a causal rejection principle, which enforces that, in case of conflicts between rules, more recent rules are preferred and older rules are overridden. Some well-known proposals are presented in [4] and [2]. In particular [4] presents a complete analysis with respect to properties that an update operator should have, with the purpose of obtaining a sure and reliable update process for our agents. In this context, it is necessary to point out that when one wants to choose a theory to develop its applications is very important to know the properties that in the theory are held. It is in this sense

oarieli@mta.ac.il

that we have focused to investigate the properties that are held under our theory, and not to present properties that it doesn't hold.

In this paper, we consider similar properties to the well-known AGM postulates. We think that is necessary to reinterpret them on the context of non-monotonic reasoning via answer set programming. In addition, we pay particular attention to our view that distinguishes beliefs from justified beliefs. As a beginning, we only study Dalal's principle of irrelevance of syntax, that according to Dalal's Principle [7] of Irrelevance of Syntax, the meaning of the knowledge that results from an update must be independent of the syntax of the original knowledge, as well as independent of the syntax of the update itself. In [4] the authors analyze and interpret the AGM postulate corresponding to Dalal's principle as follows:

$$T_1 \equiv T_2 \text{ implies } S(K \odot T_1) = S(K \odot T_2).$$

Where T_1 and T_2 are any theories. By $S(P)$ we denote the collection of all answer sets of P. If $S(P) \neq \emptyset$, then P is *consistent.* " $\odot$ " is the revision operator, and understanding that equivalence means that both programs (T_1 *and* T_2) have the same answer sets. This interpretation expresses a very demanding principle of irrelevance of syntax, due to that the AGM postulates were introduced for monotonic logics. We propose to reconsider the AGM postulates [1] under our new interpretation that considers "justified beliefs" and "belief". To this aim we have introduced in [11] a new property, which we call Weak Irrelevance of Syntax, as follows:

$$\text{(WIS): } T_1 \equiv_K T_2 \text{ implies } S(K \odot T_1) = S(K \odot T_2).$$

Where $T_1 \equiv_K T_2$ has a stronger meaning than before. The intended meaning is that T_1 and T_2 have the same justifies beliefs. We show that the proposal shown in [4] for updates almost satisfies this principle. In fact we show that for programs without tautologies, this principle holds. We should point out that this property is accepted as much in belief revision as in updates, as it is shown in [4]. Also, [4] notes, however, that tautological rules in updates are, as we believe, rare in practical applications and can be eliminated easily.

Our paper is structured as follows: In section 2 we present the general syntax of clauses, we also provide the definition of answer sets for augmented logic programs as well as some background on logic, in particular on N logic. Section 3 contains the definition about update programs given in [4] and some related concepts. Next, in section 4 we introduce our principal property called WIS and introduce our main contribution in this respect. Section 5 contains some additional properties about update operator. Finally, the conclusions are drawn in section 6.

2 Background

In this section, we give some general definitions for our theory. We define our theory about logic program, which consists of rules built over a finite set $\mathcal{A}$ of propositional atoms, where these programs can only contain default negation. Later, we introduce strong negation in similar form as in [8].

oarieli@mta.ac.il

2.1 Preliminaries

Formulas are built from propositional atoms and the 0-place connectives $\top$ and $\bot$ using negation as failure (*not*) and conjunction (,). A *rule* is an expression of the form:

$$A \leftarrow B_1, ..., B_m, not\ B_{m+1}, ..., not\ B_n. \quad (1)$$

Where A and each B_i are atoms. If $B_1, ..., B_m$,*not* $B_{m+1}, ..., notB_n$ is $\top$ then we identify rule (1) with rule A. If A is $\bot$ then we identify rule (1) with a constraint. A *program* is a finite set of rules. An *Interpretation* I is a function $I : \mathcal{L} \rightarrow \{\top, \bot\}$ that assigns a truth value to each atom in the language. For a given interpretation I and a formula F we say that I is a *model* of F if $I(F) = \top$, in the usual way in classical logic (denoted as $I \models L$). Similarly I is a *model* of a program P if it is a *model* of each formula contained in P. We restrict our attention to finite logic programs. As it is shown in [3] the Gelfond-Lifschitz transformation says that for a program P and a model $N \subseteq B_P$ (B_P denotes the set of atoms that appear in P) is defined by

$$P^N = \{rule^N : rule \in P\}$$

where

$(A \leftarrow B_1, ..., B_m, not\ C_1, ..., not\ C_n)^N$ is either:

a) $A \leftarrow B_1, ..., B_m$, if $\forall j \leq n : C_j \notin N$;
b) $\top$, otherwise.

Note that P^N is always a *definite* program (i.e., a program consisting of positive atoms only). We can therefore compute its least Herbrand model (denoted as M_{P^N}) and check whether it coincides with the model N which we started with.

Definition 1. *([3]) N is a stable model of P iff N is the least model of* P^N.

2.2 Adding Strong Negation

We extend the language adding strong negation (denoted by "$\sim$"). Syntactically, the status of the strong negation operator " $\sim$ " is different from the status of the operator "*not*" the difference is the following: *not p* can be denoted by $p \rightarrow \bot$, i.e., we use "*not*" when evidence doesn't exist about p. In the same form, we use $\sim p$ when we know that p is false or it doesn't happen. We can say that answer sets are usually defined for logic programs possessing this second kind of negation, that as we mentioned previously expresses the direct or explicit falsity of an atom.

A literal, L, is either an atom A (a positive literal) or a strongly negated atom $\sim A$ (a negative literal). For a literal L, the *complementary literal*, $\sim L$, is $\sim A$ if $L = A$, and A if $L = \sim A$, for some atom A. For a set S of literals, we define $\sim S = \{\sim L \mid L \in S\}$, and denote by $Lit_{\mathcal{A}}$ the set $\mathcal{A} \cup \sim\mathcal{A}$ for all literals over $\mathcal{A}$. A literal preceded *not* is called a *weakly negated literal*.

Therefore, a rule is an expression of the form:

oarieli@mta.ac.il

$$A \leftarrow B_1, ..., B_m, not\ B_{m+1}, ..., not\ B_n. \quad (2)$$

Where A and each B_i are literals. An Extended Logic Programs (ELPs) P is a set of rules (2). For a rule r of the form (2) we define $H(r) = \{A\}$ and $B(r) = \{B_1, ..., B_m, not\ B_{m+1}, ..., not\ B_n\}$.
The concept of modeling ($I \models P$) is extended to include strong negation as explained in [4]. The concept of answer set can be extended in a similar way, see [6]. We will consider only ELPs, in the rest of the paper unless stated otherwise.

2.3 Nelson Logics

Now, we give a brief description about N logic, because this gives an alternative interpretation of theoretical foundation to ASP [8]. Recall that a natural deduction system for intuitionistic logic can be obtained from the corresponding classical system by dropping the law of the excluded middle

$$F \vee \neg F$$

from the list of postulates. N is the minimal extension of Intuitionistic logic with strong negation and axioms of Nelson logic. A formalization of N_2 can be obtained from intuitionistic logic by adding the axiom schema $F \vee (F \rightarrow G) \vee \neg G$ and axioms of Nelson logic.

The correspondence between the language of logic programs and the language of propositional formulas in the presence of two negations is described in [8]. The main theorem in [8] readily generalizes to the new setting: we can show that two extended programs are strongly equivalent if and only if they are equivalent in the N_2 logic [8].

Is well–known that provability in N can be reduced to provability in intuitionistic logic [12]. For example, if we want to know if $\sim(a \wedge b) \rightarrow \neg b$ is a theorem in N, we can just check if $((a' \rightarrow \neg a) \wedge (b' \rightarrow \neg b)) \rightarrow ((a' \vee b') \rightarrow \neg b)$ is a theorem in I. By abuse of notation we can understand $\sim a$ as a new atom and we only ask if $((\sim a \rightarrow \neg a) \wedge (\sim b \rightarrow \neg b)) \rightarrow (\sim a \vee \sim b) \rightarrow \neg b \vdash_I b$ in our example. Similarly, provability in N_2 can be reduced to provability in G_3. We will assume this result freely in the rest of the paper.

The next theorem is a simple corollary of results in [8]. However notice that we do not require strong equivalence. Hence we just need N logic and not the full power of N_2.

Theorem 1. *For any P_1 and P_2 programs, $P_1 \equiv_N P_2$ implies that for every P program, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets.*

We will write $P \vdash_K \alpha$ to denote the fact that $P \vdash_N \alpha$. The idea for using K instead of N is due to two reasons: First, to emphasize the reading P "knows" α. Second, because strictly speaking we are translating the connective symbols. Similarly we will write $P_1 \equiv_K P_2$ instead of $P_1 \equiv_N P_2$. We will write $P_1 \equiv P_2$ to denote that P_1 and P_2 have the same answer sets.

oarieli@mta.ac.il

3 Update Programs

In [4] the authors define an *update sequence*, P as a series of two programs (P_1, P_2) of extended logic programs (ELPs). We say that P is an update sequence over $\mathcal{A}$ iff $\mathcal{A}$ represents the set of atoms occurring in the rules of the constituting elements P_i of P ($1 \leq \mathrm{i} \leq 2$). Giving an update sequence $P = (P_1, P_2)$ over $\mathcal{A}$, we assume a set $\mathcal{A}^*$ extending $\mathcal{A}$ by new, pairwise distinct atoms $rej(r)$ and A_i, for each r occurring in P, each atom $A \in \mathcal{A}$, and each i, $1 \leq i \leq 2$. We further assume an injective naming function $N(\cdot, \cdot)$, which assigns to each rule r in a program P_i a distinguished name, $N(r, P_i)$, obeying the condition $N(r, P_i) \neq N(r', P_j)$ whenever $i \neq j$. With a slight abuse of notation we shall identify r with $N(r, P_i)$ as usual. Finally, for a literal L, we write L_i to denote the result of replacing the atomic formula A of L by A_i. Let us consider the definition about the update sequence given by Eiter et al. but only in the case of two programs and let us make a slight change of notation. Also, our proposal can be extended to general case $(P_1, P_2, ..., P_n)$ in the iterative form as shown in [4]. Under certain conditions, which exclude possibilities for local inconsistencies, the iterativity property holds.

Definition 2. *[4] Giving an update of two programs $\boldsymbol{P}_{\oplus} = (P_1, P_2)$ over a set of atoms $\mathcal{A}$, we define the update program $\boldsymbol{P}_{\oplus} = P_1 \oplus P_2$ over $\mathcal{A}^*$ consisting of the following items:*

(i) all constraints in $P_1 \cup P_2$;
(ii) for each $r \in P_1$;
$L_1 \leftarrow B(r)$, not $rej(r)$. if $H(r) = L$;
$rej(r) \leftarrow B(r), \neg L_2$.
(iii) for each $r \in P_2$;
$L_2 \leftarrow B(r)$. if $H(r) = L$;
(iv) for each literal L occurring in $\boldsymbol{P}$;
$L_1 \leftarrow L_2 \qquad L \leftarrow L_1$.

Definition 3. *[4] Let $\boldsymbol{P} = (P_1, P_2)$ be an update sequence over a set of atoms $\mathcal{A}$. Then, $S \subseteq Lit_{\mathcal{A}}$ is an update answer set of $\boldsymbol{P}$ iff $S = S' \cap \mathcal{A}$ for some answer set S' of $\boldsymbol{P}_{\oplus}$. The collection of all update answer sets of $\boldsymbol{P}$ is denoted by $\mathcal{U}(\boldsymbol{P})$.*

Given two update programs P_1, P_2, we write $P_1 \equiv P_2$ if they have the same update answer sets.

Example 1. This example was taken of [4].

Let P be:
sleep ← *not* tv-on.
night.
tv-on.
watch-tv ← tv-on.

let P_1 be
∼tv-on ← power-failure.
power-failure.

Applying definition 2 to both programs, we obtain:
The single answer set of $\mathbf{P} = (P, P_1)$ using definition 3 is,
S = {power-failure, ∼tv-on, sleep, night}, as desired,
since the only answer set of $P_{\oplus}$ is given by:
{sleep1, night1, rej-r3, ∼tv-on2, power-failure, power-failure2, ∼tv-on1, power-failure1, sleep, night, ∼tv-on}

oarieli@mta.ac.il

Definition 4. *[4] Let us call two rules r_1 and r_2 conflicting iff $H(r_1) = \sim H(r_2)$.*

Definition 5. *[4] For an update sequence $\boldsymbol{P} = (P, ..., P_n)$ over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$ based on the principle of founded rule rejection, we define the rejection set of S by $Rej(S, \boldsymbol{P}) = \cup_{i=1}^{n} Rej(S, \boldsymbol{P})$, where $Rej_n(S, \boldsymbol{P}) = \emptyset$, and, for $n > i \geq 1$,*

$$Rej_i(S, \boldsymbol{P}) = \{r \in P_i \mid \exists r' \in P_j \backslash Rej_j(S, \boldsymbol{P}), \text{ for some } j \in \{i+1, ..., n\} \text{ such that } r, r' \text{ are conflicting and } S \models B(r) \cup B(r')\}.$$

Definition 6. *[4] For an update sequence $\boldsymbol{P} = (P, ..., P_n)$ over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$, let us define*

$$Rej'(S, \boldsymbol{P}) = \cup_{i=1}^{n} \{r \in P_i \mid \exists r' \in P_j, \text{ for some } j \in \{i+1, ..., n\} \text{ such that } r, r' \text{ are conflicting and } S \models B(r) \cup B(r')\}.$$

Note: Observe that Rej and Rej' coincide for two programs. Furthermore $Rej'(S, (P_1, P_2)) = \{r \in P_1 \mid \exists r' \in P_2$, such that r and r' are conflicting rules, and $S \models B(r) \cup B(r')\}$.

Lemma 1. *Let $\boldsymbol{P} = (P_1, P_2)$ be an update sequence over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$ a set of literals. Then, S is an answer set of $\boldsymbol{P}$ iff S is an answer set of $(P_1 \cup P_2 \setminus Rej'(S, P))^S$.*

Proof: S is an answer set of P iff
S is the minimal model of $(\cup P \setminus Rej(S, P))^S$ (by theorem 4 given in [4]) iff
S is an answer set of $(P_1 \cup P_2 \setminus Rej(S, P))$ iff
S is an answer set of $(P_1 \cup P_2 \setminus Rej'(S, P))$ as desired.

It is necesary to point out that in [4], update programs do not satisfy many of the properties defined in the literature. This is partly explained by the nonmonotonicity of logic programs and the causal rejection principle embodied in the semantics, which strongly depends on the syntax of rules.

Next, we present an example where the "equivalence" between two programs, P_1 and P_2 is not enough to preserve the equivalence when we update each one of these programs with another program P.

Let $P_1 =$ { a ←b. } , let $P_2 =$ { a ← a. , b ← b. } and let $P =$ {b.}

Here, P_1 and P_2 are two equivalent programs, but $P_1 \oplus P \equiv P_2 \oplus P$ is *false*. The unique answer set of P_1 and P_2 is the empty set. The only answer set of $P_1 \oplus P$ is $\{a, b\}$, on the other hand, the only answer set of $P_2 \oplus P$ is $\{b\}$.

Note that $P \oplus P_1 \equiv P \oplus P_2$ is $false$, too.

4 Weak Irrelevance of Syntax

Within our main results, we can see that the proposal presented in [4] satisfies WIS considering programs without tautologies.

oarieli@mta.ac.il

Example 2. This example shows how WIS fails.

Let P be	Let P_1 be	Let P_2 be
∼d.	h.	h.
d←h.	d←d.	h←d.

Here, P_1 and P_2 are strongly equivalent. However, when we update $P \oplus P_1$, it has only an answer set {h, d} and the update $P \oplus P_2$ doesn't have answer sets, therefore WIS fails,.

4.1 Main Results

We are now going to define a new framework for logic program updates that satisfies Weak Irrelevance of Syntax.

Definition 7. *P is tau-free w.r.t. a signature L if no rule of the form $l \leftarrow l,\alpha$ belongs to P, where α could be empty.*

Lemma 2. *Let P a program and a a literal. Suppose $P \vdash_K a$, and $M \models P$. Then $\exists\ r \in P$, where r has the form $a \leftarrow \beta$ such that $M \models \beta$.*

Lemma 3. *Let P_2, $\{c\}$ and $\{r\}$ be tau-free programs, where $r \in P_1$, and M an interpretation. Suppose $P_2 \vdash_K c$, $M \models B(r) \wedge B(c)$, and $M \models P_2$. Also, r and c are conflicting rules. Then $\exists\ r' \in P_2 \mid r'$ is a conflicting rule with r and $M \models B(r) \wedge B(r')$.*

Proof: Let c and r be formulas of the form $x \leftarrow \theta$ and $\sim x \leftarrow \beta$ respectively, also $r \in P_1$. So, $M \models B(r)$, and c and r are conflicting rules. Furthermore, $M \models P_2 \cup \{\theta\}$ (i.e., $P_2 \cup \{\theta\}$ is consistent). By the deduction theorem $P_2 \cup \{\theta\} \vdash_K x$. By lemma 2: $\exists\ r' \in P_2 \cup \{\theta\}$ such that r' is of the form $x \leftarrow B(r')$ and $M \models B(r')$. Moreover $r' \in P_2$, due to the restriction that considers only tau-free programs. So, we know that r and r' are conflicting rules and as $M \models B(r')$ then $M \models B(r) \wedge B(r')$.

Lemma 4. *Let c be a tau-free rule. Let P_1 and P_2 be tau-free programs. Suppose S is an answer set of (P_1, P_2) and $P_2 \vdash_K c$. Then $Rej(S, (P_1, P_2)) = Rej(S, (P_1, P_2 \cup \{c\})$.*

Proof: *We need to proof two cases:*
Case 1) $Rej(S, (P_1, P_2)) \subseteq Rej(S, (P_1, P_2 \cup \{c\})$. Clearly holds.

Case 2) $Rej(S, (P_1, P_2 \cup \{c\}) \subseteq Rej(S, (P_1, P_2))$

Let $r \in Rej(S, (P_1, P_2 \cup \{c\}))$ then $\exists r' \in (P_2 \cup \{c\})$ such that $S \models B(r) \wedge B(r')$. Here, we have two cases:
a) If $r' \in P_2$ then $r \in Rej(S, (P_1, P_2))$, as desired.
b) If $r' = c$, we have $r \in P_1$, $P_2 \vdash_K c$, $S \models B(r) \wedge B(c)$, r and c are conflicting rules, and $S \models P_2$ because S is an answer set of (P_1, P_2). Then, by lemma 3, $\exists r" \in P_2 \mid r$ and $r"$ are conflicting rules and $S \models B(r) \wedge B(r")$. Hence, $r \in Rej(S, (P_1, P_2))$, as desired.

oarieli@mta.ac.il

The following two notes are necessary for Lemma 5.
Note a: Rej' only erases clauses from P_1 (not from P_2).
Note b: $P' \models_K c$ implies $P' \equiv_K P' \cup \{c\}$.

Lemma 5. *Let P_1, P_2 and $\{c\}$ be tau-free programs. If $P_2 \vdash_K c$ then $P_1 \oplus P_2 \equiv P_1 \oplus (P_2 \cup \{c\})$.*

Proof:

S is an answer set of $P_1 \oplus P_2$ iff	by lemma 1
S is an answer set of $(P_1 \cup P_2) \setminus Rej'(S,(P_1,P_2))$ iff	by note a
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2))) \cup P_2$ iff	by lemma 4
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2 \cup \{c\}))) \cup P_2$ iff	by note b
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2 \cup \{c\}))) \cup (P_2 \cup \{c\})$ iff	by note a
S is an answer set of $(P_1 \cup (P_2 \cup \{c\}) \setminus Rej'(S,(P_1,P_2 \cup \{c\}))$ iff	by lemma 1
S is an answer set of $P_1 \oplus (P_2 \cup \{c\})$	

Hence $P_1 \oplus P_2 \equiv P_1 \oplus (P_2 \cup \{c\})$ as desired.

Lemma 6. *Let P, P_1 and R be tau-free programs. Suppose, that $P_1 \vdash_K R$. Then $P \oplus P_1 \equiv P \oplus (P_1 \cup R)$.*

Proof: By induction on the size of R.

Base case: Let $R = \emptyset$, then the result is immediate.
Induction Hypothesis: Let P_1, R, and $\{c\}$ be tau-free programs. We need to show: if $P_1 \vdash_K R \cup \{c\}$ then $P \oplus P_1 \equiv P \oplus (P_1 \cup (R \cup \{c\}))$.

But, we know that $P_1 \vdash_K R \cup \{c\}$ means that $P_1 \vdash_K R$ and $P_1 \vdash_K c$ then applying induction hypothesis $P \oplus P_1 \equiv P \oplus (P_1 \cup R)$
(I) By lemma 5, $P_1 \cup R \vdash_K c$ then $P \oplus (P_1 \cup R) \equiv P \oplus ((P_1 \cup R) \cup \{c\})$ (II)

Now, from (I) and (II) we have $P \oplus P_1 \equiv P \oplus ((P_1 \cup R) \cup \{c\})$
Since $P \oplus ((P_1 \cup R) \cup \{c\}) \equiv P \oplus (P_1 \cup (R \cup \{c\}))$ we obtain
$P \oplus P_1 \equiv P \oplus (P_1 \cup (R \cup \{c\}))$ as desired.

Theorem 2. *Let P, P_1 and P_2 be tau-free programs under the same language $\mathcal{L}$, if $P_1 \equiv_K P_2$ then $P \oplus P_1 \equiv P \oplus P_2$*

Proof: i) $P \oplus P_1 \equiv P \oplus (P_1 \cup P_2)$ applying lemma 6
Besides, if $P_1 \equiv_K P_2$ then $P_2 \equiv_K P_1$ and applying lemma 6 to $P \oplus P_2$ we have
ii) $P \oplus P_2 \equiv P \oplus (P_2 \cup P_1)$
Also, $P \oplus (P_1 \cup P_2) = P \oplus (P_2 \cup P_1)$
Finally, by transitivity between i) and ii) we have $P \oplus P_1 \equiv P \oplus P_2$ as desired.

It is necessary to stand out that only lemma 3 depends on the properties of the operator. Therefore, we can say that any operator satisfying lemma 3 would have the WIS property, of course, assuming the answer set semantics. Next, we present an example that satisfies WIS into our new framework.

oarieli@mta.ac.il

Example 3. This example shows that WIS holds.

Let P be:	Let P_1 be:	Let P_2 be:
∼a.	a←b.	a←b.
b.	d←a.	d←a.
		d←b.

Here, P_1 and P_2 are strongly equivalent. If we update $P \oplus P_1$ we obtain that the answer set is {b, d, a} and the update $P \oplus P_2$ has the same answer set, therefore WIS holds.

5 New Properties of the Update Operator $\oplus$

As we have mentioned, the interpretation given in [4] of the AGM postulates expresses a very demanding principle of irrelevance of syntax, because of the AGM postulates were introduced for monotonic logics. After having revised several proposals about update programs such as [2,4,1], we have some interesting properties of the style of the AGM postulates for update programs, but in our context of answer set semantics that consider the two notions: *Belief* and *Knowledge*. We call them BK-ASP properties.

Definition 8 (BK-ASP).
K1: $P \oplus x$ *is a program.*
K2: $P \oplus x \vdash_K x$.
K3: $x \equiv_K \bot$ *implies* $(P \oplus x) \equiv_K \bot$.
K4: $P \cup x \vdash_K P \oplus x$.
K5: if $P_1 \equiv_K P_2$ *then* $P \oplus P_1 \equiv P \oplus P_2$. *(WIS)*
K6: if $P_1 \vdash_K R$ *then* $(P \oplus P_1) \cup R \equiv P \oplus (P_1 \cup R)$.

We consider that the result of update is a program, and a program is considered as a theory, as we mention in the introduction. As we can see, our second (*K2*) property guarantees that the input sentence x is accepted in $P \oplus x$. With respect to our third property (*K3*), it says that if x is inconsistent (at the knowledge level) then $(P \oplus x)$ can not be consistent. With respect to our fourth property (*K4*), it says that an expansion always knows more (or equal) than an update. Our fifth (*K5*) property says that update should be analyzed on the knowledge level and not on the syntactic level. For this reason, two logically equivalent sentences (at the knowledge level) should lead to equivalent updates (at the belief level). This is the most interesting property in our proposal and it is resolved and supported by our theorem 2. Next, we present our main result with respect to BK-ASP properties.

Theorem 3. *The update operator* $\oplus$ *satisfies the six BK-ASP properties for tau-free programs.*

Proof: Our first four properties follow directly by construction. The fifth property follows by theorem 2. The last property can be proven as follows:
Under the assumption $P_1 \vdash_K R$ is easy to check that $P \oplus P_1 \equiv (P \oplus P_1) \cup R$, by lemma 6 $(P \oplus P_1) \cup R \equiv P \oplus (P_1 \cup R)$, as desired.

oarieli@mta.ac.il

6 Conclusions

We studied new properties of the update operator. Different from other approaches we considered a view of answer sets based on Nelson logics. This allowed us to reconsider the AGM postulates in a more solid framework. Our main contribution is the proposal of six properties for an update operator and the proof that $\oplus$ satisfies all of them. However, we should continue working in this line, since there are properties such as the following:

$$P \oplus P_1 \oplus Q \equiv P \oplus P_2 \oplus Q$$

for every program P and Q, that are not always satisfied, due to our property is just satisfied by the right.

References

1. C.E. Alchourron, P. Gardenfors, and D. Makinson. On the logic of Theory Change, Partial Meet Functions for Contraction and Revision Functions. *Journal of Symbolic Logic*, vol. 50 pp. 510–530, 1985.
2. J. J. Alferes, L. M. Pereira. Logic Programming Updating – a guided approach, in: A. Kakas, F. Sadri (eds.), Computational Logic: From Logic Programming into the Future – Essays in honour of Robert Kowalski, volume 2, pp. 382–412, Springer LNAI 2408, 2002.
3. G. Brewka, J. Dix, and K. Knonolige. Nonmonotonic Reasoning: An overview. CSLI Publication Eds. Leland Stanford Junior University, 1997.
4. T. Eiter, M. Fink, G. Sabattini, and H. Thompits. Considerations on Updates of Logic Programs. In M.O. Aciego, L.P. de Guzmn, G. Brewka, and L.M. Pereira, editors, Proc. Seventh European Workshop on Logic in Artificial Intelligence JELIA 2000, vol. 1919 in LNAI, Springer 2000.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programs. Proceedings of the Fifth International Conference on Logic Programming 2 MIT Press. Cambridge, Ma. pp.1070–1080.
6. M. Gelfond and V. Lifschitz. Classical negation in logic programs and Disjunctive databases. New Generation Computing. pp. 365–387, 1991.
7. H. Katsuno and A.O. Mendelzon. Propositional knowledge base revision and minimal change, Artificial Intelligence vol. 52, pp. 263–294, Elsevier, 1991.
8. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. ACM Transactions on Computational Logic. vol. 2:526–541, 2001.
9. M. Osorio, J.A. Navarro, and J. Arrazola. Equivalence in Answer Set Programming (extended version), Proceedings of LOPSTR 01, LNCS 2372, pp.57–75, Springer-Verlag, Paphos, Cyprus, November 2001.
10. M. Osorio, J.A. Navarro, and J. Arrazola. Applications of Intuitionistic Logic in Answer Set Programming, accepted in Journal of TPLP , 2003.
11. M. Osorio. and F. Zacarías. *"Irrelevance of Syntax in updating answer set programs"*, to appear in Workshop on Logic and Agents into Proc. of Fourth Mexican International Conference on Computer Science (ENC' 03) Apizaco Tlaxcala, México 2003.
12. E. Rasiowa. An algebraic approach to non–classical logics. American Elsevier Publishing company, INC. New–York, 1974.

oarieli@mta.ac.il

Minimal Keys in Higher-Order Datamodels

Attila Sali

Alfréd Rényi Institute of Mathematics
Hungarian Academy of Sciences
Budapest, P.O.B.127 H-1364 Hungary
sali@renyi.hu

Abstract. We study keys in higher-order datamodels. We show that they are equivalent with certain ideals. Based on that we introduce an ordering between key sets, and investigate systems of minimal keys. We give a sufficient condition for a Sperner-family of SHL-ideals being system of minimal keys, and give lower and upper bounds for the size of the smallest Armstrong-instance.

1 Introduction

The relational datamodel gave rise to theoretical research in several directions. Dependency structures were investigated as first-order logical sentences that are supposed to hold for all database instances [3,19]. On the other hand, their combinatorial investigations were fruitful resulting in nice problems, concepts, even as far topics as design and coding theory [8,9,10,5].

The relational model has been extended or generalized to nested relational model [15], object oriented models [16], and object-relational models. The important structures of all these were captured by the higher-order Entity-Relationship model [17,18]. The semi-structured data and XML treated in [1] can also be considered as an object-oriented model.

The major new structure in all these models is the introduction of constructors that allow us to form complex data values from simpler ones. The dependecies of the relation model can be generalized to these higher-order models, and the axiomatization of certain dependencies was carried out in [11,12,13,14]. On the other hand, the induced combinatorial structures have not been investigated thoroughly yet. It is important from the point of view of schema design, to identify what kind of attributes can form *key systems*. The aim of the present paper is to take the first steps in that direction, thus generalizing the work of [6,7, 8]. In Section 2 the neccessary definitions are recalled. In Section 3 the fundamental concept of SHL-ideals is investigated. Section 4 contains results on the combinatorial structures.

2 Higher-Order Datamodel

In this section we recall the basic definitions of the higher-order datamodel. Because of the obvious space limitations we restrict ourselves to the very neccessary facts. Since the present paper can be considered as a continuation of [14]

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 242–251, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

from the present volume, our notations follow [14]. Let a finite *universe* $\mathcal{U}$ be given, with countably infinite *domains* $dom(A)$ for all $A \in \mathcal{U}$. The elements of $\mathcal{U}$ are called *simple attributes*. Furthermore, let $\mathcal{L}$ be a set of labels, such that $\mathcal{U} \cap \mathcal{L} = \emptyset$. Let $\lambda \notin \mathcal{U} \cup \mathcal{L}$.

Definition 2.1. *The set* $\mathcal{N}$ *of* nested attributes *is the smallest set with* $\lambda \in \mathcal{N}$, $\mathcal{U} \subseteq \mathcal{N}$, *and satisfying the following properties:*

- *for* $X \in \mathcal{L}$ *and* $X'_1, \ldots, X'_n \in \mathcal{N}$ *we have* $X(X'_1, \ldots, X'_n) \in \mathcal{N}$;
- *for* $X \in \mathcal{L}$ *and* $X' \in \mathcal{N}$ *we have* $X\{X'\} \in \mathcal{N}$;
- *for* $X_1, \ldots, X_n \in \mathcal{L}$ *and* $X'_1, \ldots, X'_n \in \mathcal{N}$ *we have* $X_1(X'_1) \oplus \ldots \oplus X_n(X'_n) \in \mathcal{N}$.

λ *is called* null attribute, $X(X'_1, \ldots, X'_n)$ record attribute, $X\{X'\}$ set attribute, *and* $X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)$ (disjoint) union attribute.

The domains are extended from simple attributes to nested attributes.

Definition 2.2. *For a nested attribute* $X \in \mathcal{N}$ *a domain is assigned as follows:*

- $dom(\lambda) = \{\top\}$;
- $dom(X(X'_1,\ldots,X'_n)) = \{(X_1 : v_1,\ldots,X_n : v_n) \mid v_i \in dom(X'_i)\ i = 1,\ldots,n\}$ *with labels* X_i *for te attributes* X'_i;
- $dom(X\{X'\}) = \bigcup_{n=0}^{\infty} \{\{v_1, \ldots, v_n\} \mid v_i \in dom(X')\ i = 1, \ldots, n\}$, *i.e., each element of* $dom(X\{X'\})$ *is a finite set with elements in* $dom(X')$; *item-*$dom(X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)) = \{(X_i : v_i) \mid v_i \in dom(X'_i)\ i = 1, \ldots, n\}$.

Certain *restructuring rules* were introduced in [2,14]. The first few state that λ in record attributes can be added or removed, order does not count in record or union attributes, and applying the same constructor to equivavlent attributes results in equivalent attributes, again. The last three are there beacuse the union constructor allows partitioning the instance into subsets containing only elements of a particular label.

Definition 2.3. $\equiv$ *is the equivalence relation generated by the following rules:*

- $\lambda \equiv X()$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_1, \ldots, X'_n, \lambda)$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_{\sigma(1)}, \ldots, X'_{\sigma(n)})$ *for any permutation* σ;
- $X_1(X'_1) \oplus \ldots \oplus X_n(X'_n) \equiv X_1(X'_{\sigma(1)}) \oplus \ldots \oplus X_n(X'_{\sigma(n)})$*for any permutation* σ;
- $X(X'_1, \ldots, X'_n) \equiv X(Y_1, \ldots, Y_n)$ *iff* $X'_i \equiv Y_i$ *for all* $i = 1, \ldots, n$;
- $X_1(X'_1) \oplus \ldots \oplus X_n(X'_n) \equiv X_1(Y_1) \oplus \ldots \oplus X_n(Y_n)$ *iff* $X'_i \equiv Y_i$ *for all* $i = 1, \ldots, n$;
- $X\{X'\} \equiv X\{Y\}$ *iff* $X' \equiv Y$;
- $X(X'_1, \ldots, Y_1(Y'_1) \oplus \ldots \oplus Y_m(Y'_m), \ldots, X'_n) \equiv Y_1(X'_1, \ldots, Y'_1, \ldots, X'_n) \oplus \ldots \oplus Y_m(X'_1, \ldots, Y'_m, \ldots, X'_n)$;
- $X\{X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)\} \equiv X(X_1\{X'_1\}, \ldots, X_n\{X'_n\})$;

oarieli@mta.ac.il

In the rest of the paper $\mathcal{N}$ is identified with the set of equivalence classes $\mathcal{N}/\equiv$. An ordering is introduced between nested attributes, that is between equivalence classes of nested attributes. In the following we will write $=$ instead of $\equiv$, and we say that Y is a subattribute of X if $\hat{Y} \leq \hat{X}$ holds for some representatives $\hat{Y}$ and $\hat{X}$ of the equivalence classes of Y and X, respectively.

Definition 2.4. *For $X, Y \in \mathcal{N}$ Y is called a* subattribute *of X, iff $Y \leq X$ holds, where the partial order $\leq$ is generated by the following rules:*

- $X \geq \lambda$ *for all* $X \in \mathcal{N}$;
- $X(Y_1, \ldots, Y_n) \geq X(X'_{\sigma(1)}, \ldots, X'_{\sigma(m)})$ *for some injective* $\sigma\colon \{1, \ldots, m\} \to \{1, \ldots, n\}$ *and* $Y_{\sigma(i)} \geq X'_{\sigma(i)}$ *for* $i = 1, \ldots, m$;
- $X_1(Y_1) \oplus \ldots \oplus X_n(Y_n) \geq X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)$ *if* $Y_i \geq X'_i$ *for* $i = 1, \ldots, n$;
- $X\{X'\} \geq X\{Y\}$ *iff* $X' \geq Y$;
- $X(X_{i_1}\{\lambda\}, \ldots, X_{i_k}\{\lambda\}) \geq X_{\{i_1, \ldots, i_k\}}\{\lambda\}$;

The last case of Definition 2.4 requires some explanation. Since $X\{X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)\}$ is equivalent with $X(X_1\{X'_1\}, \ldots, X_n\{X'_n\})$, subattributes of type $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})$ occur in $\mathcal{S}(X)$ for every subset $I = \{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$. On the other hand, these latter subattributes are equivalent with $X\{X_{i_1}(X'_{i_1}) \oplus \ldots \oplus X_{i_k}(X'_{i_k})\}$, thus we obtain subattributes of type "$X\{\lambda\}$" for all subsets I of the indices. These subattributes need to be distinguished from each other, which is done by introducing new labels X_I and writing $X_I\{\lambda\}$. For example, $X_\emptyset\{\lambda\} = \lambda$, $X_{\{1,2,\ldots,n\}}\{\lambda\} = X\{\lambda\}$ and $X_{\{i\}}\{\lambda\} = X(X_i\{\lambda\})$. (A tuple t has $\pi^X_{X_I\{\lambda\}} = \top$ iff it has a nonempty element of label X_i for some $i \in I$.)

For a nested attribute $X \in \mathcal{N}$, $\mathcal{S}(X)$ denotes the poset of subattributes under the ordering given in Definition 2.4. This poset is a lattice, but not neccessarily distributive [14]. If $Y \leq X$, then there is a *natural projection* $\pi^X_Y\colon dom(X) \to dom(Y)$. If $X \equiv Y$, then we have both, $X \leq Y$ and $X \geq Y$ and the projections π^X_Y and π^Y_X are inverses of each other.

A *weak functional dependency* on $\mathcal{S}(X)$ is an expression $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with I being a (finite) index set and $\mathcal{Y}_i, \mathcal{Z} \subseteq \mathcal{S}(X)$. If $|I| = 1$, a *functional dependency* is obtained. A finite subset $r \subseteq dom(X)$ is called an *an instance* of X. r is said to *satisfy* the weak functional dependency $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ (notation $r \models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$) iff for every pair of tuples $t_1, t_2 \in r$ there exists an $i \in I$ such that $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}_i$ implies $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}_i$. Note, that if $|I| = 1$, we obtain the usual concept of satisfying a functional dependency.

We need one more definition.

Definition 2.5. *Two subattributes $Y, Z \in \mathcal{S}(X)$ are called* semi-disjoint *iff one of the following holds:*

1. $Y \geq Z$ *or* $Z \geq Y$;
2. $X = X(X_1, \ldots, X_n)$, $Y = X(Y_1, \ldots, Y_n)$, $Z = X(Z_1, \ldots, Z_n)$ *and* $Y_i, Z_i \in \mathcal{S}(X_i)$ *are semi-disjoint for all* $i = 1, \ldots, n$;
3. $X = X_1(X'_1) \oplus \ldots \oplus X_n(X'_n)$, $Y = X_1(Y'_1) \oplus \ldots \oplus X_n(Y'_n)$, $Z = X_1(Z'_1) \oplus \ldots \oplus X_n(Z'_n)$, *and* $Y'_i, Z'_i \in \mathcal{S}(X'_i)$ *are semi-disjoint for all* $i = 1, \ldots, n$;

oarieli@mta.ac.il

3 SHL Ideals

In this section we study an important concept of the theory of higher order datamodels. This is an ideal of $\mathcal{S}(X)$ with additional closure properties, which was introduced in [14].

Definition 3.1. *Let X be a nested attribute. An* SHL-ideal *is a subset $\mathcal{F} \subseteq \mathcal{S}(X)$ with the following properties:*

1. $\lambda \in \mathcal{F}$;
2. $Y \in \mathcal{F}$ *and* $Y \geq Z$ *for* $Z \in \mathcal{S}(X)$ *implies* $Z \in \mathcal{F}$;
3. *if* $Y, Z \in \mathcal{F}$ *are semi-disjoint, then* $Y \sqcup Z \in \mathcal{F}$, *where* $\sqcup$ *is the union operator of the lattice* $\mathcal{S}(X)$;
4. a) *if* $X_I\{\lambda\} \in \mathcal{F}$ *and if* $X_J\{\lambda\} \notin \mathcal{F}$ *for* $\{i_1, i_2, \ldots, i_k\} = I \subsetneq J$, *then* $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \in \mathcal{F}$;
   b) *if* $X_I\{\lambda\} \in \mathcal{F}$ *and if* $X_i\{\lambda\} \notin \mathcal{F}$ *for all* $i \in I$, *then there exists a partition* $I = I_1 \dot\cup I_2$ *with* $X_{I_1}\{\lambda\} \notin \mathcal{F}$, $X_{I_2}\{\lambda\} \notin \mathcal{F}$ *and* $X_{I'}\{\lambda\} \in \mathcal{F}$ *for all* $I' \subseteq I$ *with* $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$;
   c) *if* $X_{\{1,\ldots,n\}} \in \mathcal{F}$ *and* $X_{I^-}\{\lambda\} \notin \mathcal{F}$ *(for* $I^- = \{i \in \{1, \ldots, n\} \mid X(X_i\{\lambda\}) \notin \mathcal{F}\}$*), then there exists some* $i \in I^+ = \{1, \ldots, n\} \setminus I^-$ *such that for all* $J \subseteq I^-$ *we have* $X_{J\cup\{i\}}\{\lambda\} \in \mathcal{F}$;
   d) *if* $X_{I^-}\{\lambda\} \in \mathcal{F}$ *and for all* $i \in I^+$ *there is some* $J \subseteq I^-$ *with* $X_{J\cup\{i\}}\{\lambda\} \notin \mathcal{F}$, *then for all* $\ell \in I^+$ *and all* $K \subseteq I^-$ *with* $X_K\{\lambda\} \notin \mathcal{F}$ *we also have* $X_{K\cup\{\ell\}}\{\lambda\} \notin \mathcal{F}$;
5. *if* $X_I\{\lambda\} \in \mathcal{F}$ *and* $X_J\{\lambda\} \in \mathcal{F}$ *with* $I \cap J = \emptyset$, *then* $X_{I\cup J}\{\lambda\} \in \mathcal{F}$;
6. a) *if* $X = X(X_1, X_2, \ldots X_n)$, *then* $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X_i) \mid X(\lambda, \ldots, Y_i, \ldots, \lambda) \in \mathcal{F}\}$ *is an SHL-ideal;*
   b) *if* $X = X_1(X'_1) \oplus \ldots \oplus X_n(x'_n)$ *and* $\mathcal{F} \neq \{\lambda\}$, *then the set* $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X_i) \mid X_1(\lambda) \oplus \ldots \oplus X_i(Y_i) \oplus \ldots \oplus X_n(\lambda) \in \mathcal{F}\}$ *is an SHL-ideal;*
   c) *if* $X = X\{X'\}$ *and* $\mathcal{F} \neq \{\lambda\}$, *then* $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X\{Y\} \in \mathcal{F}\}$ *is a semi-SHL-ideal.*

An ideal is called *semi-SHL-ideal* iff it satisfies properties 1,2,4 and the modification of 6 where we require only semi-SHL-ideal instead of SHL-ideal.

Lemma 3.1. *Let X be a nested attribute and r be an instance of it. Let $t_1, t_2 \in r$ and define $\mathcal{H}_{t_1,t_2} = \{Y \in \mathcal{S}(X) \mid \pi^X_Y(t_1) = \pi^X_Y(t_2)\}$. Then $\mathcal{H}_{t_1,t_2}$ is an SHL-ideal.*

Proof of Lemma 3.1: The first two properties are obvious, using the fact that if $Y \geq Z$ in $\mathcal{S}(X)$, then $\pi^X_Z(t) = \pi^Y_Z(\pi^X_Y(t))$ for any tuple t.
For 3., assume that $Y, Z \in \mathcal{F}$ are semi-disjoint. By Definition 2.5 either Y and Z are comparable in the lattice $\mathcal{S}(X)$, or they are constructed from semi-disjoint components via tuple or union constructors. In both of these cases easy induction completes the proof using such properties of $\mathcal{S}(X)$, as $\sqcup$ of tuples is the tuple of $\sqcup$ of the components, etc.
4.(a): $\pi^X_{X_J\{\lambda\}}(t_1) \neq \pi^X_{X_J\{\lambda\}}(t_2)$ means that one is $\emptyset$, the other is $\{\top\}$, say $\pi^X_{X_J\{\lambda\}}(t_1) = \emptyset$. This implies that $\pi^X_{X_I\{\lambda\}}(t_1) = \emptyset$, thus $\pi^X_{X_I\{\lambda\}}(t_2) = \emptyset$, as well. This means $\pi^X_{X(X_{i_1}\{X'_{i_1}\},\ldots,X_{i_k}\{X'_{i_k}\})}(t_1) = \pi^X_{X(X_{i_1}\{X'_{i_1}\},\ldots,X_{i_k}\{X'_{i_k}\})}(t_2)$.

oarieli@mta.ac.il

4.(b): Assume $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_I\{\lambda\}}(t_2)$ but $X_i\{\lambda\} \notin \mathcal{F}$, that is $\pi^X_{X_i\{\lambda\}}(t_1) \neq \pi^X_{X_i\{\lambda\}}(t_2)$ for all $i \in I$. Then for all $i \in I$ *exactly* one of t_1 and t_2 has an element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$. Let $I_0 = \{i \mid \pi^X_{X_i\{\lambda\}} = \top\}$, $I_1 = I - I_0$. Let $I' \subseteq I$, then $\pi^X_{X_{I'}\{\lambda\}}(t_1) = \pi^X_{X_{I'}\{\lambda\}}(t_2)$ iff $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$.
4.(c): $X_{\{1,\dots,n\}} \in \mathcal{F}$ implies that either both of t_1 and t_2 are the empty tuple, or both of them are non-empty. $i \in I^-$ means that *exactly* one of the two tuples have an element of label X_i. $X_{I^-}\{\lambda\} \notin \mathcal{F}$ means that t_1 and t_2 differ on the set I, that is one of them has no element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$ for $i \in I^-$, the other one has. Say, $\pi^X_{X_{I^-}}(t_1) = \emptyset$. Then there exists an $i \in I^+$, such that t_1 has an element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$. Since this index i is from I^+, t_2 must also have an element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$. In this case $\pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_1) = \pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_2) = \top$ for all $J \subseteq I^-$.
4.(d): If $X_{I^-}\{\lambda\} \in \mathcal{F}$ and for all $i \in I^+$ there is some $J \subseteq I^-$ with $X_{J\cup\{i\}}\{\lambda\} \notin \mathcal{F}$, then none of t_1 or t_2 has an element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$ for all $i \in I^+$. If $X_K\{\lambda\} \notin \mathcal{F}$ for $K \subseteq I^-$, then one of t_1 or t_2 has an element of type $(X_i : v_i)$ with $v_i \in dom(X_i)$ for some $i \in K$, while the other has none. Then if $\ell \in I^+$, then the same holds for $K \cup \{\ell\}$, as well.
5. is obvious, and to prove 6. (a)-(c), one can use induction. ■
We need here a corollary of the Central Lemma of [14].

Lemma 3.2. *Let X be a nested attribute such that the union constructor appears in X only inside a set constructor. Let $\mathfrak{H} = \{\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_k\}$ be a Sperner family of SHL-ideals. Then there exist tuples $t^i_0, t^i_1 \in dom(X)$ for all $i = 1, 2, \dots, k$, such that we have $\pi^X_Y(t^i_0) = \pi^X_Y(t^i_1)$ iff $Y \in \mathcal{H}_i$. Furthermore, if $\pi^X_Z(t^i_a) = \pi^X_Z(t^j_b)$ $a, b \in \{0, 1\}$, $i \neq j$, for some $Z \in \mathcal{S}(X)$, then Z is constructed only from subattributes of type $X_I\{\lambda\}$.*

Proof of Lemma 3.2: The Cover Lemma of [14] is applied separately for each $\mathcal{H}_i$ to obtain the tuples $t^i_0, t^i_1 \in dom(X)$. The only thing we have to take care of that during the inductive construction of the tuples, the constants from the domains of simple attributes used for $\mathcal{H}_i$ must be distinct from those used for $\mathcal{H}_j$ if $i \neq j$. This ensures that tuples constructed for $\mathcal{H}_i$ cannot agree with tuples constructed for $\mathcal{H}_j$ on subattributes having a non-λ component, for $i \neq j$. ■

4 Minimal Keys

In this section the idea of *keys* is generalized for the higher-order datamodel. In the relational model a *subset* $\mathcal{K}$ of attributes is a *key* iff $\mathcal{K} \to \mathcal{X}$. This could be interpreted in two ways if the relational model is considered as a special case of the higher-order model. That is, the *nested attribute* $X(A_1, A_2, \dots, A_n)$, where A_i's are simple attributes is equivalent with the classical relational schema $\mathcal{R}(A_1, A_2, \dots, A_n)$. A subset of attributes $\mathcal{K} = \{A_{i_1}, A_{i_2}, \dots, A_{i_r}\}$ could be identified with the *set* of subattributes $\{X(A_{i_1}), X(A_{i_2}), \dots, X(A_{i_r})\}$ or with the subattribute $X(A_{i_1}, A_{i_2}, \dots, A_{i_r})$. While the first one considers subsets of the lattice $\mathcal{S}(X)$, the second refers to elements of it. As it will be seen, the two things are different manifestations of the same general concept.

oarieli@mta.ac.il

Definition 4.1. *Let X be a nested attribute with subattribute lattice $\mathcal{S}(X)$. A subset $\mathcal{K} \subseteq \mathcal{S}(X)$ is a* key *iff $\mathcal{K} \to \mathcal{S}(X)$ holds. That is, in an* instance *of $\mathcal{S}(X)$ there exists no two tuples t_1 and t_2 such that $\pi^X_K(t_1) = \pi^X_K(t_2)$ for all $K \in \mathcal{K}$.*

Let $\mathcal{K}$ be key and consider the SHL-ideal $shl(\mathcal{K})$ generated by $\mathcal{K}$. Clearly, $shl(\mathcal{K})$ is also a key. This converse is also true.

Theorem 4.1. *Let $\mathcal{H}$ be an SHL-ideal of $\mathcal{S}(X)$, which is a key. Let $\mathcal{G} \subset \mathcal{H}$ be any generating subset of $\mathcal{H}$. Then $\mathcal{G} \to \mathcal{S}(X)$ also holds, i.e., $\mathcal{G}$ is also a key.*

Proof of Theorem 4.1: The proof of Theorem 5.1 of [14] is an applied. Consider the one-element set of FDs $\Sigma = \{\mathcal{H} \to \mathcal{S}(X)\}$, and assume that $\{\mathcal{G} \to \mathcal{S}(X)\} \notin \Sigma^+$. That is, there exists a $Z \in \mathcal{S}(X)$, such that $\{\mathcal{G} \to \{Z\}\} \notin \Sigma^+$. Let $\mathcal{Z}$ be the filter generated by Z, i.e., $\mathcal{Z} = \{Y \mid Y \geq Z\}$ and let $\mathcal{U} = \mathcal{S}(X) - \mathcal{G} - \mathcal{Z}$. Clearly, $\mathcal{G} \cap \mathcal{Z} = \emptyset$. Let r be an instance such that $r \not\models \{\mathcal{G} \to \{Z\}\} \iff r \not\models \{\mathcal{G} \to \{Y\} \mid Y \in \mathcal{Z}\}$. That is, there exists $t_1, t_2 \in r$ such that $\pi^X_G(t_1) = \pi^X_G(t_2)$ for all $G \in \mathcal{G}$, but $\pi^X_Y(t_1) \neq \pi^X_Y(t_2)$ for all $Y \in \mathcal{Z}$. Take a maximal $\mathcal{U}' \subseteq \mathcal{U}$ such that $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}'$. If $r \models \{\mathcal{G} \cup \mathcal{U}' \to \{Y\} \mid Y \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}$ held, then $\mathcal{U}' \neq \mathcal{U}$ would hold, and there would exist some $V \in \mathcal{U} - \mathcal{U}'$ with $\pi^X_V(t_1) = \pi^X_V(t_2)$, which contradicts the maximality of $\mathcal{U}'$. Thus, there exists $\mathcal{U}' \subseteq \mathcal{U}$ such that $r \not\models \{\mathcal{G} \cup \mathcal{U}' \to \{Y\} \mid Y \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}$. Take $\mathcal{U}'$ maximal with respect to this property. Then $\mathcal{I} = \mathcal{G} \cup \mathcal{U}'$ is an SHL-ideal, as it is shown in [14]. However, $shl(\mathcal{G}) = \mathcal{K} \subseteq \mathcal{I}$, so $\mathcal{I}$ is a key, in particular $r \models \{\mathcal{G} \cup \mathcal{U}' \to \{Y\} \mid Y \in \mathcal{S}(X)\}$, that contradicts the choice of $\mathcal{U}'$. ∎

Theorem 4.1 allows us to identify keys with their generated SHL-ideals and systems of keys with families of SHL-ideals. Two keys are said to be *equivalent*, if they generate the *same* SHL-ideal. We introduce an ordering on the (equivalence classes of) keys as follows

$$\mathcal{K}_1 \preceq \mathcal{K}_2 \iff shl(\mathcal{K}_1) \subseteq shl(\mathcal{K}_2). \tag{1}$$

A key is said to be *minimal*, if it is minimal under the ordering $\preceq$. Thus, a system $\mathfrak{K}$ of minimal keys corresponds to a *Sperner*-family $\mathfrak{H}$ of SHL-ideals.

Example Let $X = X(A_1, A_2, \ldots, A_n)$ be a nested attribute, where A_i's are simple attributes. Then *any* pair of subattributes are semi-disjoint, so *any* SHL-ideal is a principal ideal. The SHL-ideal generated by $\{X(A_{i_1}), X(A_{i_2}), \ldots, X(A_{i_r})\}$ happens to be the principal ideal generated by $X(A_{i_1}, A_{i_2}, \ldots, A_{i_r})$. This shows that Definition 4.1 is a sound generalization of the classical relational case.

Let $\mathcal{K}$ be a key, furthermore, let $\mathcal{H} = shl(\mathcal{K})$. The maximal elements of $\mathcal{H}$ under the ordering $\leq$ of $\mathcal{S}(X)$ form a generating set of $\mathcal{H}$, which is called the *canonical* generating set of $\mathcal{H}$. Whenever it is not stated explicitly otherwise, we will consider the canonical generating set of an SHL-ideal.

Definition 4.2. *A set $\mathcal{A} \subset \mathcal{S}(X)$ of subattributes is called an* antikey *iff $\mathcal{A} \nrightarrow \mathcal{S}(X)$.*

Using Theorem 4.1 we can prove the equivalence of antikeys and SHL-ideals.

Theorem 4.2. *Let $\mathcal{A}$ be an antikey and let $\mathcal{G} = shl(\mathcal{A})$ be the SHL-iedal generated by $\mathcal{A}$. Then $\mathcal{G}$ (and consequently any generating set of it) is an antikey.*

oarieli@mta.ac.il

PROOF OF THEOREM 4.2: Assume indirectly that $\mathcal{A}$ is an antikey, but $\mathcal{G} = shl(\mathcal{A})$ is not, that is, $shl(\mathcal{A})$ is a key. Then applying Theorem 4.1 we obtain that *any* generating set of $shl(\mathcal{A})$, in particular $\mathcal{A}$ also, is a key, a contradiction. ■

The ordering $\preceq$ defined by (1) can be extended to (equivalence classes of) antikeys, as well. The *maximum* antikeys form a Sperner-family $\mathfrak{A}$ of SHL-ideals. Let $\mathfrak{K} = \{\mathcal{K}_1, \mathcal{K}_2, \dots, \mathcal{K}_r\}$ be the system of minimal keys. Then the set of maximum antikeys is $\mathfrak{A} = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_s\}$ so that

$$\mathcal{K}_i \not\subseteq \mathcal{A}_j \text{ for all pairs } i, j \text{ and the } \mathcal{A}_j\text{'s are maximal under this condition.} \quad (2)$$

Let $\mathfrak{K} = \{\mathcal{K}_1, \mathcal{K}_2, \dots, \mathcal{K}_r\}$ be a Sperner-system of SHL-ideals, furthermore let $\mathfrak{A} = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_s\}$ be the Sperner-system of SHL-ideals defined by (4.2). In this case $\mathfrak{A}$ is denoted by $\mathfrak{A} = \mathfrak{K}^{-1}$.

The question is *which Sperner-systems of SHL-ideals occur as families of minimal keys*? In the relational model the answer was given by Armstrong and Demetrovics [4,6]. Namely they proved that *every* Sperner-family of subsets of attributes can be a system of minimal keys. However, the analogous statement does not hold in general for the higher order datamodel. Consider the nested attribute $X\{X_1(A) \oplus X_2(B)\} = X(X_1\{A\}, X_2\{B\})$ and its subattribute lattice $\mathcal{S}(X)$ shown in Figure 1. Let us take $\mathfrak{K} = \{\{\{X\{\lambda\}\}, \{\lambda\}\}\}$, that is the system

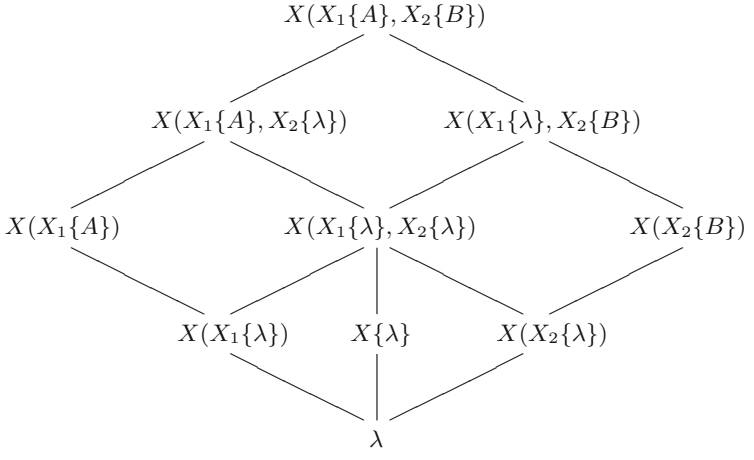


Fig. 1. The lattice $\mathcal{S}(X\{X_1(A) \oplus X_2(B)\})$

of minimal keys consisiting of a single SHL-ideal generated by $\{X\{\lambda\}\}$. Beacuse $\{X\{\lambda\}\}$ is a key, there are only two possible tuples in an instance, namely $t_1 = \emptyset$ and t_2 being any nonempty tuple. The system of maximal antikeys $\mathfrak{A}$

oarieli@mta.ac.il

consists of two SHL-ideals, $\mathcal{A}_1 = shl(X(X_1\{A\}))$ and $\mathcal{A}_2 = shl(X(X_2\{B\}))$. By Lemma 4.1 we would need at least 3 tuples to realize this antikey system.

Definition 4.3. *Let $\mathfrak{K} = \{\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_r\}$ be a Sperner-system of SHL-ideals. r is called an* Armstrong-instance *for $\mathfrak{K}$ if the minimal key system determined by r is $\mathfrak{K}$. If there exists an Armstrong instance for a given $\mathfrak{K}$, then $s(\mathfrak{K})$ denotes the smallest possible number of tuples in an Armstrong-instance of $\mathfrak{K}$. Otherwise define $s(\mathfrak{K}) = \infty$.*

The next lemma is a generalization of the key lemma from [7].

Lemma 4.1. *Let $\mathfrak{K} = \{\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_r\}$ be a Sperner-system of SHL-ideals. Let $\mathfrak{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s\}$ be the system of maximal antikeys belonging to $\mathfrak{K}$. Then*

$$\binom{s(\mathfrak{K})}{2} \geq |\mathfrak{A}| \tag{3}$$

Proof of Lemma 4.1: If $s(\mathfrak{K}) = \infty$, then we have nothing to prove. Otherwise, let r be an instance of minimum number of tuples. For $t_i \neq t_j \in r$ let $\mathcal{H}_{ij}$ be the SHL-ideal of subattributes where the two tuples agree by Lemma 3.1. This is an antikey, so there exists an $\mathcal{A}_k$ with $\mathcal{H}_{ij} \subseteq \mathcal{A}_k$. On the other hand, $\bigcup_{1\leq i<j\leq |r|} \mathcal{H}_{ij} \supseteq \bigcup_{1\leq k\leq |\mathfrak{A}|} \mathcal{A}_k$ must hold. Thus, the number of pairs of tuples in r is at least as large as the number of maximal antikeys. ■

Consider $X = X_1(X_1') \oplus X_2(X_2') \oplus \ldots \oplus X_n(X_n')$. An instance r of X can be partitioned $r = r_1 \dot\cup r_2 \dot\cup \ldots \dot\cup r_n$ where r_i consists of the tuples of type $(X_i : v_i)$. A set $\mathcal{K} = \{Y_1, Y_2, \ldots, Y_m\} \subseteq \mathcal{S}(X)$ is a key iff $\pi^X_{X_i(X_i')}(\mathcal{K})$ is a key in $\mathcal{S}(X_i)$ for all $1 \leq i \leq n$. Note that if $Y_j = X_1(Y_1^j) \oplus \ldots \oplus X_n(Y_n^j)$, then $\pi^X_{X_i(X_i')}(\mathcal{K}) = \{X_i(Y_i^1), X_i(Y_i^2), \ldots, X_i(Y_i^m)\}$. This observation allows us to consider only the cases when the union constructor occours only inside of a set, list or multiset constructor.

Theorem 4.3. *Let X be a nested attribute and assume that the union constructor occours in X only inside of a set constructor. Let $\mathfrak{K} = \{\mathcal{K}_1, \mathcal{K}_2, \ldots \mathcal{K}_n\}$ be a Sperner-family of SHL-ideals of $\mathcal{S}(X)$. If the canonical generating set of $\mathcal{K}_i$ contains a subattribute that has a simple attribute in its construction, for all $1 \leq i \leq n$, then $\mathfrak{K}$ has an Armstrong-instance and $s(\mathfrak{K}) \leq 2|\mathfrak{K}^{-1}|$.*

Proof of Theorem 4.3: Let $\mathfrak{K}^{-1} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$. Lemma 3.2 provides tuples t_0^i, t_1^i $i = 1, 2, \ldots, k$ such that $\pi^X_Y(t_0^i) = \pi^X_Y(t_1^i)$ iff $Y \in \mathcal{A}_i$. This ensures that $\mathcal{A}_i$ is an antikey for all i. On the other hand, if $\pi^X_Z(t_a^i) = \pi^X_Z(t_b^j)$ for some $Z \in \mathcal{S}(X)$, $a, b \in \{0, 1\}$, and $1 \leq i < j \leq k$, then Z is constructed only from subattributes of type $X_I\{\lambda\}$, thus no two tuples can agree on every subattribute in the canonical generating set of $\mathcal{K}_s$ for all s, which implies that each $\mathcal{K}_s$ is a key. The number of tuples in this Armstrong-instance is exactly $2|\mathfrak{K}^{-1}|$. ■

The condition in Theorem 4.3 is sufficient, but not neccessary for the existence of Armstrong-instance. For example, consider $\mathcal{S}(X\{X_1(A) \oplus X_2(B)\})$ shown on Figure 1. If we take $\mathfrak{K} = \{shl(\{X(X_1\{\lambda\}, X_2\{\lambda\})\})\}$, then $\mathfrak{K}^{-1}$ consists of two

oarieli@mta.ac.il

SHL-ideals, $\mathcal{A}_1 = shl(X(X_1\{A\}))$ and $\mathcal{A}_2 = shl(X(X_2\{B\}))$. Then three tuples are constructed in Lemma 3.2, namely $t_0^1 = t_0^2 = \emptyset$, $t_1^1 = (X_2 : b)$, and $t_1^2 = (X_1 : a)$. These three tuples clearly form an Armstrong-instance for $\mathfrak{K}$.

5 Conclusion

In the present paper we investigated keys and antikeys in the presence of various constructors in the higher order datamodel. We proved that keys, as well as antikeys, correspond to certain ideals with additional closure properties. These are SHL-ideals, subsets of the subattribute lattice. We showed that the system of minimal keys correspond to Sperner-system of SHL-ideals and exhibited a sufficient condition when such a Sperner-system occurs as a system of minimal keys. The candidate key systems not covered by the sufficient condition of Theorem 4.3 are the patological cases in the sense that having a key SHL-ideal whose generating sets are constructed of subattributes of type $X_I\{\lambda\}$ entirely limits the possible number of tuples to a finite value. These types are not likely to occure in practice.

Demetrovics [6] used his construction to determine the maximum possible number of minimal keys in a relational database schema. Our results may turn out to be useful to answer the similar problem in the presence of various constructors of the higher order datamodel. The SHL-ideals of the subattribute lattice form a lattice themselves under the set containment as ordering. We conjecture that the largest Sperner-system of this latter lattice has an Armstrong-instance (true for the relational model).

References

1. S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From relations to Semistructured Data and XML*, Morgan Kaufmann Publishers, 2000.
2. S. Abiteboul, and R. Hull, Restructuring hierarchical database objects, *Theoretical Computer Science* (1988)
3. S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.
4. W.W. Armstrong, Dependency structures of data base relationships, *information Processing* (1974), 580–583.
5. G. Brightwell and G.O.H. Katona, A new type of coding theorem, *Studia Sci. Math. Hungar.* **38** (2001) 139–147.
6. J. Demetrovics, On the equivalence of candidate keys with Sperner systems, *Acta Cybernetica (Szeged)* **4** (1979) 247–252.
7. J. Demetrovics and G.O.H. Katona, Extremal combinatorial problems in relational data base, *Lecture Notes in Computer Science* **117** Springer, Berlin 1981, 110–119.
8. J. Demetrovics, G.O.H. Katona, and A. Sali, The characterization of branching dependencies, *Discrete Applied Mathematics*, **40** (1992) 139–153.
9. J. Demetrovics, G.O.H. Katona, and A. Sali, Design type problems motivated by database theory, *Journal of Statistical Planning and Inference*, **72**, (1998), 149–164.
10. B. Ganter, H.-D.O.F. Gronau and R.C. Mullin, On orthogonal double covers of K_n, *Ars Combinatoria*, **37**, (1994) 209–221.

oarieli@mta.ac.il

11. S.Hartmann and S. Link, Reasoning about functional dependencies in an abstract data model, *Electronic Notes in Theoretical Computer Science 84* (2003)
12. S.Hartmann, S. Link, and K.-D. Schewe, Capturing functional dependencies in the higher-order antity relationship model. Tech. Rep. 5/2003, Massey University, Department of Information Systems, (2003)
13. S.Hartmann, S. Link, and K.-D. Schewe, Reasoning about functional and multivalued dependencies in the presence of base, record and finite list types, submitted, 2003.
14. S.Hartmann, S. Link, and K.-D. Schewe, *Weak Functional Dependencies in Higher-Order Datamodels – The case of the union constructor –*, in the same volume.
15. J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht, *The Structure of the Relational Database Model*, Springer-Verlag, 1989.
16. K.-D. Schewe and B. Thalheim, Fundamental concepts of object oriented databases, *Acta Cybernetica* **11** 4 (1993), 49–85.
17. B. Thalheim, Foundations of entity-relationship modeling, *Annals of Mathematics and Artificial Intelligence* **6** (1992) 197–256.
18. B. Thalheim, *Entity-Relationship Modeling: Foundations of Database Technology*, Springer-Verlag, 2000.
19. J.D. Ullman, *Principles of Database Systems*, Computer Science Press, Inc., Rockville, MD: 1982

oarieli@mta.ac.il

Similarity Relational Calculus and Its Reduction to a Similarity Algebra

Ingo Schmitt and Nadine Schulz

Otto-von-Guericke University Magdeburg
Institute of Technical and Business Information Systems
Universitätsplatz 2, 39106 Magdeburg, Germany
{schmitt|nschulz}@iti.cs.uni-magdeburg.de

Abstract. Traditional database query languages are based on set theory and crisp logic. Many applications, however, need similarity or retrieval-like queries producing results with truth values from the interval $[0, 1]$. Such truth values can be regarded as continuous membership values of tuples expressing how strongly a query is matched. Formulating queries by applying existing similarity relational algebras means to express the user's need in a procedural manner. In order to support a declarative way of formulating queries, we generalize the classical relational domain calculus by incorporating fuzzy operations and user weights. Besides defining syntax and semantics we show how to map any calculus expression onto a corresponding similarity algebra expression. In this way, we present a theoretical foundation for a declarative query language combining retrieval functionality and traditional relational databases.

1 Introduction

Queries in multimedia databases often need a combination of information retrieval mechanisms and traditional database query language constructs. Retrieval functionality is required if a query contains a similarity predicate, e.g. the query: ≫*Retrieve all images that are similar in form and color to the given im lqq* In this example we have two conjunctively combined similarity predicates.

Traditional boolean operators, however, are not able to deal with imprecise membership data. Therefore, fuzzy-logic operations as a generalization of boolean logic are proposed [1]. Introducing user weights gives users more control to map the information need onto an adequate query. For example, we may formulate: ≫*Retrieve all images that are similar in form and color to the given image. And color is twice as much important as form.*≪

In order to formulate queries independently from internal query processing we propose to formulate them in a declarative manner. Therefore, we enhance the classical relational domain calculus by the notion of vagueness. For query processing, calculus expressions need to be mapped to an algebra. Therefore, our similarity algebra is the target language for the mapping but not the language which the user poses queries against. Since querying directly in relation calculus is too difficult for many users we are planning to design a graphical language

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 252–272, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

in a QBE-like fashion from which a mapping to our similarity calculus can be easily performed. For this reason we need a sound theoretical foundation for declarative query languages which is presented here.

Our calculus language defines core functionalities. It extends the classical relational calculus by introducing imprecise truth values in form of similarity predicates, fuzzy operators, fuzzy quantifiers, and user weights. The language is defined on the relational model, that is, imprecise data result from applying vague predicates during the query processing but not from the database directly. Since we do not specify the exact fuzzy-operations and weighting formulas our approach serves as a formal framework which can be adapted and extended to match the needs for different scenarios.

The contributions of our work can be summarized as follows:

1. We formally define a declarative query language, the similarity relational calculus, which combines the handling of imprecise truth values together with the traditional relational domain calculus.
2. The language provides operations to weight similarity predicates. Furthermore, we introduce fuzzified quantifiers.
3. We show how to map the similarity relational calculus language to a similarity algebra, because an algebra is better eligible for efficient query processing. For the mapping we distinguish between domain dependent and domain independent queries.

2 Related Work

The relational data model and its languages, the relational algebra and the relational calculus, were developed by Codd and published in [2,3,4,5]. He proved the equivalence between algebra and calculus by specifying a reduction algorithm. A good overview of the theory of relational databases is given in [6,7,8].

An important aspect of the reduction algorithm is to restrict calculus queries so that they produce finite and domain independent results only. Such queries are called safe queries. A discussion concerning the safety aspect regarding our similarity calculus and algebra is given in [9].

In our approach we enhance the relational domain calculus by vague predicates. Coping with vagueness requires appropriate logical operators. Therefore, we apply techniques from fuzzy-logic [1]. Furthermore, we give users freedom to specify preferences for operands of operators. Weights on operands express their weighted contribution to the operation result. Typically, an approach as described by Fagin and Wimmers in [10] can be applied. However, we leave the weighting mechanism open. That is, our approach is not restricted to Fagin's formula. In [11] we discussed the application of Fagin's formula to complex similarity queries w.r.t. associativity and distributivity.

Another approach to specify preferences is introduced in [12]. Instead of weighting preferences in terms of $\gg A$ *is better than* $B \ll$ are map onto strict partial orders. Beyond it, a weighting of search terms can be indirectly expressed

oarieli@mta.ac.il

using certain predicates which requires the user to specify appropriate combing functions.

In the early nineties, much research was done on developing fuzzy-databases with corresponding fuzzy query languages. [13] introduces a fuzzy ER-model together with a calculus language using Fuzzy-logic. Another example is [14] which investigates how to implement a Fuzzy-SQL language on top of the commercial database system ORACLE. Most of the work in the area of fuzzy databases, however, do not support user weights. Furthermore, they rely on the two imprecise values *necessity* and *possibility* which do not conform to our intended scenario of multimedia applications.

[15] sketches the design of a fuzzy calculus, fuzzy algebra and a mapping between them. However, this work suffers from an incomplete formalization.

Most extensions of the relational model by imprecision were performed on the relational algebra, see e.g. [16,17]. A very good work is [17] which introduced the $same^w$ similarity algebra. Our proposal defines a small set of algebra operations which is powerful enough to be the target language for mapping from the similarity calculus. One problem, not considered in [17], is the observation that a weighted conjunction where the score of one operand is zero can produce a nonzero score. Furthermore, in contrast to [17] we leave the semantics of unweighted und weighted operators unspecified and require just the satisfaction of some logical rules.

An interesting approach to combine the information retrieval world with the database world is the probability relational algebra proposed in [18]. However, due to the stochastic approach they require stochastically independent events (tuples). Therefore, the authors propose intensional semantics instead of extensional semantics which is typically used in the database area. A problem with this approach occurs when an imprecise predicate violates the demand for independent events (tuples).

3 Weighting Fuzzy Operations

In general a calculus query consists of a condition X, that is composed of n predicates x_i with $i = 1 \ldots n$. A complex query condition is a compound of predicates using the operators $\wedge$ and $\vee$. Beside these connectives the negation operator is likewise important[1]. Thus, a query condition can be defined as $X := x \mid (X\ [\wedge|\vee]\ X) \mid \neg X \mid (X)$.

Similarity or retrieval-like predicates produce values from the interval $[0, 1]$ called scores. The overall tuple score μ based on an aggregation (conjunction, disjunction) of the specific truth values μ_i is calculated using a scoring function $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$. Typically, t-norms resp. t-conorms from Fuzzy-Logic [1] are employed as scoring-functions for conjunctions resp. disjunctions. These functions must hold the following conditions:

[1] In this section quantifiers are neglected.

oarieli@mta.ac.il

Definition 3.1. *Function* $T : [0,1] \times [0,1] \to [0,1]$ *is a t-norm if it satisfies the criteria:*

(i)	$T(\mu_1, \mu_2) = T(\mu_2, \mu_1)$	*(commutativity)*
(ii)	$T(\mu_1, T(\mu_2, \mu_3)) = T(T(\mu_1, \mu_2), \mu_3)$	*(associativity)*
(iii)	$\mu_1 \leq \mu_2 \wedge \mu_3 \leq \mu_4 \Rightarrow T(\mu_1, \mu_2) \leq T(\mu_3, \mu_4)$	*(monotonicity)*
(iv)	$T(\mu_1, 1) = \mu_1$	*(border condition).*

Definition 3.2. *Function* $T : [0,1] \times [0,1] \to [0,1]$ *is a t-conorm if it satisfies the criteria:*

(i)-(iii)	*same as in the definition of t-norms*	
(iv)	$T(\mu_1, 0) = \mu_1$	*(border condition).*

An aspect to enhance the flexibility to express preferences is to give users the possibility to assign weights to arguments in a compound query [17,10,19]. In our approach we support a binary weighting of search terms by incorporating a weight $\theta \in [0,1]$ into the classical operators $\wedge$ and $\vee$. We distinguish between left-oriented ($\overleftarrow{\wedge}_\theta, \overleftarrow{\vee}_\theta$) and right-oriented ($\overrightarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta$) operators. The arrow marks the side on which the weight is stronger. Thus, $\overrightarrow{\wedge}_\theta$ denotes that the right operand is stronger weighted than the left one and $\overleftarrow{\wedge}_\theta$ denotes that the left operand is stronger weighted than right one, respectively. We require $\mu_1 \overleftarrow{\wedge}_\theta \mu_2 = \mu_1$ if $\theta = 1$ and $\mu_1 \overleftarrow{\wedge}_\theta \mu_2 = \mu_1 \wedge \mu_2$ if $\theta = 0$. The same must be hold by $\overrightarrow{\wedge}_\theta$, $\overleftarrow{\vee}_\theta$, and $\overrightarrow{\vee}_\theta$.

For the evaluation of weighted conjunctions and disjunctions it is necessary to incorporate the weight θ into underlying scoring functions. Therefore, in our approach the signature of a weighted scoring function, for example for $\overleftarrow{\wedge}_\theta$, is: $S^{\Theta}_{\overleftarrow{\wedge}_\theta} : [0,1] \times [0,1] \times [0,1] \to [0,1]$, where the first argument is given by the weight.

There exist several requirements for a weighted scoring function [10,19,20]. We require a weighted scoring function to be a generalization of the corresponding, unweighted scoring function. Furthermore, the weighted function should be continuous in all its arguments and in case of equal weights or equal input scores it should reduce to the unweighted scoring function.

Fagin and Wimmers proposed a weighting formula which allows to incorporate weights into any scoring function [10]. The weighting formula requires the weights to be given for each operand. Thus, a mapping of the θ value from $\overrightarrow{\wedge}_\theta$ to θ_1, θ_2 for Fagin's formula is performed by applying $\theta_1 = \frac{1-\theta}{2}$ and $\theta_2 = \frac{1+\theta}{2}$. The same holds for $\overrightarrow{\vee}_\theta$. For the left-arrowed cases the formulas for the weights θ_1 and θ_2 need to be swapped. Fagin's approach is applicable for n-ary scoring functions. Under the assumption of $\sum_i \theta_i = 1$ and $\theta_1 \geq \ldots \geq \theta_n$ the weighting formula is:

$$S^{\Theta}(\mu_1, \ldots, \mu_n, \theta_1, \ldots, \theta_n) = (\theta_1 - \theta_2) S(\mu_1) + 2 * (\theta_2 - \theta_3) S(\mu_1, \mu_2) + \ldots + n * \theta_n S(\mu_1, \ldots, \mu_n)$$

oarieli@mta.ac.il

where S^{Θ} denotes the weighted scoring function based on the unweighted function S, the weights θ_i, and the scores μ_i with $i = 1, \ldots, n$. There exist further weighted scoring functions employed in various retrieval systems, see e. g. in [21, 22]. Nevertheless, we recommend to use Fagin's weighting formula to obtain weighted scoring functions since these meet most of our requirements.

One problem occurs if Fagin's formula is applied to a scoring function without idempotence. In case of equal input scores but different weights the formula does not reduce to the unweighted case. Assume, for example, the binary t-norm is the algebraic product $\mu_1 * \mu_2$ and $\mu_1 = \mu_2$ holds then Fagin's weighting produces

$$(\theta_1 - \theta_2) * \mu_1 + 2 * \theta_2 * \mu_1 * \mu_1 = (1 - 2\theta_2) * \mu_1 + 2 * \theta_2 * \mu_1 * \mu_1 \neq \mu_1 * \mu_1.$$

We suggest a small modification of Fagin's formula:

$$S^{\Theta}(\mu_1, \mu_2, \theta_1, \theta_2) = (\theta_1 - \theta_2) S(\mu_1, \mu_1) + 2 * \theta_2 S(\mu_1, \mu_2)$$

to solve this problem.

4 Similarity Calculus

In this section we formally define the syntax and semantics of our similarity calculus language. The design is based on the following principles:

1. The language is a generalization of the relational domain calculus. Thus, every traditional relational domain query can be expressed and evaluated in our language producing the same query result.
2. Fuzzy truth values are generated primarily by applying similarity predicates.
3. The result of a query is a relation which contains all tuples with a non-zero truth value.
4. Our language definitions provide an open framework for different scenarios with corresponding similarity predicates. That is, we exactly define the semantics of many language constructs, but leave the semantics of similarity predicates open. In this way, our language is open to cope, for example, with histogram intersections as a special similarity predicate. This predicate is required for measuring image similarity upon color distribution.
5. User weights on operators are expressed by weight variables. Their values are fixed outside the query by an interpretation function.
6. We introduce fuzzyfied quantifiers which soften the strict semantics of traditional quantifiers.

4.1 Syntax

We start by defining the basic syntax elements of the language.

Definition 4.1. *We denote the* similarity domain relation calculus $\mathcal{SDC}$ *as a tuple* $(\boldsymbol{U}, \boldsymbol{X}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$*, where* $\boldsymbol{U} = \{A_1, A_2, \ldots\}$ *is the universe of attributes;* $\boldsymbol{X} = \{X_1, X_2, \ldots\}$ *is a set of variables;* $\Delta = \{\delta_1, \delta_2, \ldots\}$ *is a set*

oarieli@mta.ac.il

of binary, typed[2] *operation names (we distinguish continuous operations, e. g. similarity operations, from discrete operations, e. g. traditional comparison operators like* $<, \leq, =, \neq, >, \geq$*);* $\boldsymbol{C}$ *is a set of constant names;* $\boldsymbol{D}$ *is a set of domain names; Dom is a mapping from* $\boldsymbol{U} \cup \boldsymbol{X} \cup \Delta \cup \boldsymbol{C}$ *to* $\boldsymbol{D}$*;* $\boldsymbol{R}$ *is a finite set of relation schemata* $R_1, R_2, \ldots, R_p$*, all are subsets*[3] *of* $\boldsymbol{U}$*; and* $\Theta = \{\theta_1, \theta_2, \ldots\}$ *is a set of weight variables.*

We build a calculus query expression E over $\mathcal{SDC}$ from *atoms* and *formulas.*

Definition 4.2. *Let an* atom *be*

1. $R(Y_1, Y_2, \ldots, Y_m)$*, where* $R \in \boldsymbol{R}$ *is a relation schema* $A_1, A_2, \ldots, A_m$ *and for each* $Y_i \in \boldsymbol{X} \cup \boldsymbol{C}$ *the domain is sound:* $Dom(Y_i) = Dom(A_i)$.
2. $Y_1 \delta Y_2$*, where* $\delta \in \Delta$ *is an operation name (infix notation) and* $Y_1, Y_2 \in \boldsymbol{X} \cup \boldsymbol{C}$ *are consistently typed* ($Dom(Y_1) = Dom(Y_2) = Dom(\delta)$).

Please note, that both operands of a binary operation can be constants at the same time. In this case, interpreting such an atom results in a truth value independent from any variable.

Definition 4.3. *An* $\mathcal{SDC}$-formula $F(X_1, X_2, \ldots, X_n)$[4]*, where* $X_i \in \boldsymbol{X}, i = 1, \ldots, n$ *are the involved free variables, is recursively defined:*

1. *Any atom is a formula* $F(X_1, X_2, \ldots, X_n)$ *where* X_i *are the involved variables.*
2. $(F_a(X_{a_1}, \ldots, X_{a_k}) \phi F_b(X_{b_1}, \ldots, X_{b_l}))$ *with* $\phi \in \{\wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta\}$ *is a formula*
   $F(X_1, \ldots, X_n)$ *if* $F_a(X_{a_1}, \ldots, X_{a_k})$ *as well as* $F_b(X_{b_1}, \ldots, X_{b_l})$ *are formulas. Involved variables are united:* $\{X_1, \ldots, X_n\} = \{X_{a_1}, \ldots, X_{a_k}\} \cup \{X_{b_1}, \ldots, X_{b_l}\}$.
   In case of using weighted operators ($\overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta$)*,* $\theta \in \Theta$ *is a weight variable and* $\overrightarrow{\wedge}_\theta$ *resp.* $\overrightarrow{\vee}_\theta$ *denotes that* F_b *is stronger weighted than* F_a *and* $\overleftarrow{\wedge}_\theta$ *resp.* $\overleftarrow{\vee}_\theta$ *denotes that* F_a *is stronger weighted than* F_b*, respectively.*
3. $(\neg F(X_1, \ldots, X_n))$ *is a formula if* $F(X_1, \ldots, X_n)$ *is a formula.*
4. $(\exists X F(X_1, \ldots, X_n))$ *is a formula if* $F(X_1, \ldots, X_n)$ *is a formula and* $X \in \{X_1, \ldots, X_n\}$.
5. $(\forall X F(X_1, \ldots, X_n))$ *is a formula if* $F(X_1, \ldots, X_n)$ *is a formula and* $X \in \{X_1, \ldots, X_n\}$.
6. $(\exists_k X F(X_1, \ldots, X_n))$ *is a formula if* $F(X_1, \ldots, X_n)$ *is a formula,* $k > 1$ *is a natural number, and* $X \in \{X_1, \ldots, X_n\}$.
7. $(\forall_k X F(X_1, \ldots, X_n))$ *is a formula if* $F(X_1, \ldots, X_n)$ *is a formula,* $k > 1$ *is a natural number, and* $X \in \{X_1, \ldots, X_n\}$.

[2] Both operands are assumed to be from the same domain.
[3] For convenience, we assume a fixed attribute order.
[4] As short form we often write F instead of $F(X_1, X_2, \ldots, X_n)$.

oarieli@mta.ac.il

In our language we distinguish between left-oriented ($\overleftarrow{\wedge}_\theta$, $\overleftarrow{\vee}_\theta$) and right-oriented ($\overrightarrow{\wedge}_\theta$, $\overrightarrow{\vee}_\theta$) weighted operators. This is necessary because a weighted conjunction can produce a membership value of non-zero although one operand equals zero. The reason is the requirement for a weighting formula, that if a weight is completely on one side, the other operand should be completely ignored. In order to reason over weighted operators we introduce, therefore, asymmetric weighted operators.

The normal exists- and forall-quantifier are sometimes too strict because their results often depend on one single value. Therefore, we introduce the fuzzyfied quantifiers $\exists_k X$ and $\forall_k X$, also called *few* and *most*. In our approach, these quantifiers need at least k different significant X-values to behave like the normal $\exists$ and $\forall$, respectively. Otherwise, the quantifiers provide only a corresponding fraction of the normal quantifier value. The formal definition is given in Definition 4.8.

Definition 4.4. *A* query expression E *over* $\mathcal{SDC}$ *has the form*

$$\{X_1, X_2, \ldots, X_n | F(X_1, X_2, \ldots, X_n)\}$$

where $F(X_1, X_2, \ldots, X_n)$ *is an SDC-formula and* $X_1, \ldots, X_n$ *are the involved free variables.*

The query asks for the values for all variables where the condition F holds.

In order to guarantee finite and domain independent results we require safe queries. To verify safety syntactically we adopt and extend the approach given in [23]. Due to space restrictions, we do not discuss this aspect here and refer to [9].

4.2 Semantics

After defining the syntax of our $\mathcal{SDC}$ language we specify the semantics of $\mathcal{SDC}$-expressions. Therefore, we first give the interpretation over $\mathcal{SDC}$ in Definition 4.5 followed by definitions for variable mapping and evaluating atoms. We specify the semantics of an $\mathcal{SDC}$-formula in Definition 4.8 and, finally, we define the semantics of an $\mathcal{SDC}$-query expression.

Definition 4.5. *An* interpretation *over* $\mathcal{SDC}(\boldsymbol{U}, \boldsymbol{X}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *is a triple* $(\boldsymbol{d}, \boldsymbol{db}, I)$*, where*

1. $\boldsymbol{d}$ *is a finite set of domains* $\{d_1, d_2, \ldots, d_q\}$*, each domain is a non-empty, not necessarily finite set of values and* $\boldsymbol{db}$ *is a finite set of finite relations* $\{r_1, r_2, \ldots, r_p\}$ *over these domains.*
2. I *is an interpretation function which*
   a) *maps any domain name* $D \in \boldsymbol{D}$ *to a domain* $I(D) \in \boldsymbol{d}$*,*
   b) *maps any relation schema* $R(A_1, A_2, \ldots, A_n) \in \boldsymbol{R}$ *to a relation* $I(R) \in \boldsymbol{db}$ *where* $I(R) \subseteq I(Dom(A_1)) \times I(Dom(A_2)) \times \ldots \times I(Dom(A_n))$*,*
   c) *maps any operation name* $\delta \in \Delta$ *to a binary function:* $I(\delta) : I(Dom(\delta)) \times I(Dom(\delta)) \to [0,1]$.[5] *By convention, there exists an*

[5] Discrete operation values are restricted to $\{0, 1\}$ where 0 denotes `false` and 1 denotes `true`.

oarieli@mta.ac.il

operation ‚=‘ with the equality-semantics which can be applied to every datatype.

d) maps any constant name $C \in \boldsymbol{C}$ *to a value* $I(C) \in I(Dom(C))$,

e) maps any weight variable $\theta \in \boldsymbol{\Theta}$ *to a value* $I(\theta) \in [0,1]$,

f) maps the conjunction symbol ‘$\wedge$*’ to a fuzzy t-norm* $I(\wedge) : [0,1] \times [0,1] \to [0,1]$. *It must hold:* $\forall \mu \in [0,1] : I(\wedge)(0,\mu) = I(\wedge)(\mu,0) = 0$. *Since ‘*$\wedge$*’ is associative and commutative we generalize it to an n-ary operator.*

g) maps the weighted conjunction symbol ‘$\overrightarrow{\wedge}_\theta$*’ to a weighted fuzzy t-norm* $I(\overrightarrow{\wedge}_\theta) : [0,1] \times [0,1] \times [0,1] \to [0,1]$. *The first parameter is reserved for the weight* $I(\theta)$. *Furthermore, the following condition must hold:* $\forall v, \mu \in [0,1], I(\overrightarrow{\wedge}_\theta)(v,\mu,0) = 0$. *The value 0 for* θ *produces the unweighted conjunction whereas the value 1 ignores the less weighted operand:*

$$\forall \mu_1, \mu_2 \in [0,1] : I(\overrightarrow{\wedge}_\theta)(0,\mu_1,\mu_2) = I(\wedge)(\mu_1,\mu_2)$$
$$\forall \mu_1, \mu_2 \in [0,1] : I(\overrightarrow{\wedge}_\theta)(1,\mu_1,\mu_2) = \mu_2.$$

Furthermore, equal input scores are reduced to the unweighted case: $\forall \theta, \mu \in [0,1] : I(\overrightarrow{\wedge}_\theta)(\theta,\mu,\mu) = I(\wedge)(\mu,\mu)$

Mapping the weighted conjunction symbol ‘$\overleftarrow{\wedge}_\theta$*’ is analogously.*

h) maps the disjunction symbol ‘$\vee$*’ to a fuzzy t-conorm* $I(\vee) : [0,1] \times [0,1] \to [0,1]$. *It must hold:* $\forall \mu \in [0,1] : I(\vee)(1,\mu) = I(\vee)(\mu,1) = 1$. *Since ‘*$\vee$*’ is associative and commutative we generalize it to an n-ary operator.*

i) maps the weighted disjunction symbol ‘$\overrightarrow{\vee}_\theta$*’ to a weighted fuzzy t-conorm* $I(\overrightarrow{\vee}_\theta) : [0,1] \times [0,1] \times [0,1] \to [0,1]$. *The first parameter is reserved for the weight* $I(\theta)$. *Furthermore, the following condition must hold:* $\forall v, \mu \in [0,1], I(\overrightarrow{\vee}_\theta)(v,\mu,1) = 1$. *The value 0 for* θ *produces the unweighted disjunction whereas the value 1 ignores the less weighted operand:*

$$\forall \mu_1, \mu_2 \in [0,1] : I(\overrightarrow{\vee}_\theta)(0,\mu_1,\mu_2) = I(\vee)(\mu_1,\mu_2)$$
$$\forall \mu_1, \mu_2 \in [0,1] : I(\overrightarrow{\vee}_\theta)(1,\mu_1,\mu_2) = \mu_2.$$

Furthermore, equal input scores are reduced to the unweighted case: $\forall \theta, \mu \in [0,1] : I(\overrightarrow{\vee}_\theta)(\theta,\mu,\mu) = I(\vee)(\mu,\mu)$.

Mapping the weighted disjunction symbol ‘$\overleftarrow{\vee}_\theta$*’ is analogously.*

j) maps the negation symbol ‘$\neg$*’ to a fuzzy negation:* $I(\neg) : [0,1] \to [0,1]$ *with* $\neg\neg\mu = \mu$. *Furthermore, we require the fuzzy negation to conform weighted and unweighted disjunction/conjunction w.r.t. DeMorgan’s laws.*

Please notice, the semantics of the operations $\delta, \wedge, \overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \vee, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta$ *is not predefined and can therefore be arbitrarily defined as long as the specified restrictions are met. In this way, the language is defined as a framework which works with many domain specific similarity operations and fuzzy operations.*

oarieli@mta.ac.il

The common semantics of the fuzzy-conjunction is the `min`-function, and the `max`-function for the fuzzy-disjunction. The dominant weighting formula is Fagin's formula described in [10]. In the next definition we assign a value to every variable.

Definition 4.6. *Let V be a* variable mapping *from $\boldsymbol{X}$ to $\bigcup_{d \in \boldsymbol{d}} d$ where $\forall X \in \boldsymbol{X}.V(X) \in I(Dom(X))$ holds.*

Now we can assign a value to every atom.

Definition 4.7. *The* evaluation *$V^*(F)$ of an $\mathcal{SDC}$ atom F with respect to a variable mapping V and an interpretation function I is given by:*

1. *If $F = R(Y_1, Y_2, \ldots, Y_m)$ then $V^*(F) = 1$ if $(v_1, v_2, \ldots, v_m) \in I(R)$ where*

$$i \in \{1, \ldots, m\}.v_i = \begin{cases} V(Y_i) \text{ if } Y_i \text{ is a variable} \\ I(Y_i) \text{ if } Y_i \text{ is a constant name,} \end{cases}$$

   otherwise $V^*(F) = 0$.
2. *If $F = Y_1 \delta Y_2$ then $V^*(F) = I(\delta)(v_1, v_2)$ where v_i is defined as above.*

Definition 4.8. *The* semantics *of an $\mathcal{SDC}$-formula $F(X_1, X_2, \ldots, X_n)$ denoted $I_V^*(F)$ basing on the evaluation of atoms $V^*(F)$ and an interpretation function I is recursively defined:*

1. *If F is an atom then $I_V^*(F) = V^*(F)$.*
2. *If F is a conjunction $(F_1 \wedge F_2)$ then $I_V^*(F) = I(\wedge)(I_V^*(F_1), I_V^*(F_2))$.*
3. *If F is a disjunction $(F_1 \vee F_2)$ then $I_V^*(F) = I(\vee)(I_V^*(F_1), I_V^*(F_2))$.*
4. *If F is a weighted conjunction or disjunction $(F_1 \phi F_2)$ with $\phi \in \{\overrightarrow{\wedge}_\theta, \overleftarrow{\wedge}_\theta, \overrightarrow{\vee}_\theta, \overleftarrow{\vee}_\theta\}$ then $I_V^*(F) = I(\phi)(I(\theta), I_V^*(F_1), I_V^*(F_2))$.*
5. *If F is a negation $(\neg F_1)$ then $I_V^*(F) = I(\neg)(I_V^*(F_1))$.*
6. *If F is an existentially bound formula $(\exists X F_1(X_1, \ldots, X_n))$ then $I_V^*(F) = I(\vee)(I_{\boldsymbol{V}_X}^*(F_1))$ where $\boldsymbol{V}_X = \{V_X$ is a variable mapping where $X_i \neq X$ implies $V_X(X_i) = V(X_i)$ for all $X_i \in \boldsymbol{X}\}$. The expression $I(\vee)(I_{\boldsymbol{V}_X}^*(F_1))$ denotes a disjunction over all variable mappings from $\boldsymbol{V}_X$ applied to F_1. The set $\boldsymbol{V}_X$ can be infinite. Due to distributivity and associativity, the disjunction can be applied to an arbitrary number of truth values. Applied to one value it returns exactly that value.*
7. *The $\forall$-case is analogous to the $\exists$-case except the disjunction is replaced by the conjunction.*
8. *If F is a k-existentially bound formula $(\exists_k X F_1(X_1, \ldots, X_n))$ then*

$$I_V^*(F) = I(\vee)(I_{\boldsymbol{V}_X}^*(F_1)) * \min\left(\frac{\sum_{V_X \in \boldsymbol{V}_X} I_{V_X}^*(F_1)}{k * I(\vee)(I_{\boldsymbol{V}_X}^*(F_1))}, 1\right).$$

9. *If F is a k-universally bound formula $(\forall_k X F_1(X_1, \ldots, X_n))$ then*

$$I_V^*(F) = I_V^*((\neg(\exists_k(\neg F_1(X_1, \ldots, X_n))))).$$

oarieli@mta.ac.il

Now we are able to define the semantics of a query expression. We require a non-zero truth value for every tuple of the result. Thus, the result of a query is a relation.

Definition 4.9. *The* semantics *of a query expression* $E = \{X_1, X_2, \ldots, X_n | F(X_1, X_2, \ldots, X_n)\}$ *denoted* $I^*(E)$ *is* $\{(V(X_1), V(X_2), \ldots, V(X_n)) | I_V^*(F(X_1, X_2, \ldots, X_n)) > 0\}$.

5 Similarity Algebra

In the following we introduce our similarity algebra $\mathcal{SA}$, which is the target language for mapping from $\mathcal{SDC}$ expressions. We do not consider it as a language where user formulates queries against. $\mathcal{SA}$ enhances traditional relational algebra by introducing vagueness and weighting. The similarity algebra $\mathcal{SA}$ is defined as follows:

Definition 5.1. *The tuple* $(\boldsymbol{U}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *denotes the* similarity algebra $\mathcal{SA}$*, where* $\boldsymbol{U}$*,* Δ*,* $\boldsymbol{C}$*,* $\boldsymbol{D}$*,* Dom*,* $\boldsymbol{R}$*,* Θ *have the same meaning as for the* $\mathcal{SDC}$ *language (see Definition 4.1).*

Let $att(E)$ be all attributes occurring in an algebra expression E. Every attribute is denoted by the ordinal number $\#_i$ of its occurrence in E. We now define a similarity algebra expression as given in the following definition.

Definition 5.2. *A* similarity algebra expression E *over a similarity algebra* $\mathcal{SA}$ *is the smallest class of expressions which include the following:*

1. $\boldsymbol{0}$*, that is a special relation needed for the mapping.*
2. $\boldsymbol{1}$*, that is a special relation needed for the mapping.*
3. $R \in \boldsymbol{R}$
4. Dom_D *with* $D \in \boldsymbol{D}$
5. $\pi_{\#_{p_1}, \#_{p_2}, \ldots, \#_{p_n}}(E)$*, where* E *is a similarity algebra expression and* $\{\#_{p_1}, \#_{p_2}, \ldots, \#_{p_n}\} \subseteq att(E)$.
6. $\pi^k_{\#_{p_1}, \#_{p_2}, \ldots, \#_{p_n}}(E)$*, where* E *is a similarity algebra expression,* $k > 1$ *a natural number, and* $\{\#_{p_1}, \#_{p_2}, \ldots, \#_{p_n}\} \subseteq att(E)$*. This operation is the counterpart for the* k*-exists-quantifier of the similarity calculus.*
7. $\sigma_{y_i \delta y_j}(E)$*, where* $\delta \in \Delta$ *is a binary operation and* $Dom(y_i) = Dom(y_j) = Dom(\delta)$*,* $\{y_i, y_j\} \subseteq att(E) \cup \boldsymbol{C}$*, and* E *is a similarity algebra expression. Both operands can be constants at the same time. We sometimes specify a list of conjunctively combined simple conditions as a short form of a list of corresponding selections.*
8. *unions:* $(E_1 \cup E_2)$*,* $(E_1 \overrightarrow{\cup}_\theta E_2)$*,* $(E_1 \overleftarrow{\cup}_\theta E_2)$ *where* E_1 *and* E_2 *union compatible*[6] *similarity algebra expressions.*

[6] Union compatibility means that the two expressions E_1 and E_2 must share the same number of attributes and the same domain for every corresponding attribute pair.

oarieli@mta.ac.il

9. *intersections:* $(E_1 \cap E_2)$, $(E_1 \overrightarrow{\cap}_\theta E_2)$, $(E_1 \overleftarrow{\cap}_\theta E_2)$ *where* E_1 *and* E_2 *are union compatible similarity algebra expressions.*
10. $(E_1 \setminus E_2)$*, where* E_1 *and* E_2 *are union compatible similarity algebra expressions.*
11. $(E_1 \times E_2)$*, where* E_1 *and* E_2 *are similarity algebra expressions. Since '*$\times$*' is associative and commutative we generalize it to an n-ary cartesian product.*
12. $(E_1 \bowtie_{\#_{i_1}=\#_{j_1},\ldots,\#_{i_m}=\#_{j_m}} E_2)$ *where* E_1 *and* E_2 *are similarity algebra expressions and* $\{\#_{i_1}, \ldots, \#_{i_m}\} \subseteq att(E_1), \{\#_{j_1}, \ldots, \#_{j_m}\} \subseteq att(E_2)$.

The *interpretation* of a similarity algebra $SA(\mathbf{U}, \Delta, \mathbf{C}, \mathbf{D}, Dom, \mathbf{R}, \Theta)$ except for the operations is the same as for $\mathcal{SDC}$. Therefore, we refer to Definition 4.5.

In the following, we define the semantics of an algebra expression. Notice, that we equip relations with an artificial membership attribute $\#_0$ at first attribute position. Our algebra can deal with infinite sets. Since the algebra is intended to be a target language we assume any algebra expression to be created by mapping an $\mathcal{SDC}$-query. As we will see later, any safe $\mathcal{SDC}$-query is mapped to an algebra expression basing on finite sets.

Definition 5.3. *The* semantics *of an algebra expression E is inductively defined by the interpretation function* I^**:*

1. 0-relation $E = \boldsymbol{0}$*:* $I^*(\boldsymbol{0}) = \{(1,0)\}$ *is a relation with exactly one tuple with membership value 1 and an arbitrary attribute value preferable the value 0.*
2. 1-relation $E = \boldsymbol{1}$*:* $I^*(\boldsymbol{1}) = \{(1)\}$ *is a relation with exactly one tuple with membership value 1 and no attribute value.*
3. relation name $E = R \in \mathbf{R}$*:* $I^*(R) = \{(1, v_1, v_2, \ldots, v_n) | (v_1, v_2, \ldots, v_n) \in I(R)\}$ *where* $A_1, A_2, \ldots, A_n$ *are the attributes of R. All tuples are equipped with a membership value 1 since they are considered as true facts.*
4. domain $E = Dom_D$*:* $I^*(Dom_D) = \{(1, v) | v \in I(D)\}$*. All domain values are equipped with a membership value 1 since they are considered as true facts. Please notice, that the interpretation of a domain can be an infinite set.*
5. projection $E = \pi_{\#_{p_1},\#_{p_2},\ldots,\#_{p_n}}(E_1)$*: Let* $v_{0_1}, \ldots, v_{0_l}$ *be all membership values for a fixed value list* $v_{p_1}, \ldots, v_{p_n}$ *where* $(v_{0_i}, v_1, \ldots, v_m) \in I^*(E_1)$ *holds and the corresponding values are identical:* $p_i = j \Rightarrow v_{p_i} = v_j$ *for* $i = 1, \ldots, n$*.* $I^*(\pi_{\#_{p_1},\#_{p_2},\ldots,\#_{p_n}}(E_1)) = \{(u_0, v_{p_1}, v_{p_2}, \ldots, v_{p_n}) | (v_{0_i}, v_1, \ldots, v_m) \in I^*(E_1)\}$ *where*

$$u_0 = \begin{cases} I(\vee)(v_{0_1}, \ldots, v_{0_l}) \text{ if } l > 1 \\ v_{0_1} \text{ if } l = 1 \end{cases}.$$

   Please notice, that duplicate elimination means an OR-aggregation of grouped membership values. For convenience, we apply an n-ary OR-operator since the binary OR-operator as a t-conorm holds commutativity and associativity.
6. projection $E = \pi^k_{\#_{p_1},\#_{p_2},\ldots,\#_{p_n}}(E_1)$*: Let* $v_{0_1}, \ldots, v_{0_l}$ *be all membership values for a fixed value list* $v_{p_1}, \ldots, v_{p_n}$ *where* $(v_{0_i}, v_1, \ldots, v_m) \in I^*(E_1)$ *holds, the corresponding values are identical* $p_i = j \Rightarrow v_{p_i} = v_j$ *for* $i = 1, \ldots, n$*, and*

$$u_0 = I(\vee)(v_{0_1}, \ldots, v_{0_l}) * \min\left(\frac{v_{0_1} + \ldots + v_{0_l}}{k * I(\vee)(v_{0_1}, \ldots, v_{0_l})}, 1\right).$$

oarieli@mta.ac.il

$I^*(\pi_{\#_{p_1},\#_{p_2},\ldots,\#_{p_n}}(E_1)) = \{(u_0, v_{p_1}, v_{p_2}, \ldots, v_{p_n}) | (v_{0_i}, v_1, \ldots, v_m) \in I^*(E_1)\}$

7. selection $E = \sigma_{y_i \delta y_j}(E_1)$:
$I^*(\sigma_{y_i \delta y_j}(E_1)) =$
$\{(u_0, v_1, \ldots, v_n) | (v_0, v_1, \ldots, v_n) \in I^*(E_1) \wedge u_0 = I(\wedge)(v_0, I(\delta)(\hat{y}_i, \hat{y}_j)) \wedge u_0 > 0\}$ *where*

$$\hat{y}_i = \begin{cases} v_i & \text{if } y_i \text{ is an attribute} \\ I(y_i) & \text{if } y_i \text{ is a constant name.} \end{cases}$$

$$\hat{y}_j = \begin{cases} v_j & \text{if } y_j \text{ is an attribute} \\ I(y_j) & \text{if } y_j \text{ is a constant name.} \end{cases}$$

A selection without condition means no restriction: $I^*(\sigma(E_1)) = I^*(E_1)$.

8. unions:
   - union $E = (E_1 \cup E_2) : I^*((E_1 \cup E_2)) =$
     $\{(I(\vee)(v_0, w_0), v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \ldots, v_k) \in I^*(E_2)\} \cup \{(I(\vee)(v_0, 0), v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge \forall w_0.(w_0, v_1, \ldots, v_k) \notin I^*(E_2)\} \cup \{(I(\vee)(0, w_0), v_1, \ldots, v_k) | (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge \forall v_0.(v_0, v_1, \ldots, v_k) \notin I^*(E_1)\}$.
   - weighted union $E = (E_1 \overrightarrow{\cup}_\theta E_2) : I^*((E_1 \overrightarrow{\cup}_\theta E_2)) =$
     $\{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup \{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge \forall w_0.(w_0, v_1, \ldots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup \{(u_0, v_1, \ldots, v_k) | (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge \forall v_0.(v_0, v_1, \ldots, v_k) \notin I^*(E_1) \wedge u_0 > 0\}$ *where*

$$u_0 = \begin{cases} I(\overrightarrow{\vee}_\theta)(I(\theta), v_0, w_0) & \textit{case 1} \\ I(\overrightarrow{\vee}_\theta)(I(\theta), v_0, 0) & \textit{case 2} \\ I(\overrightarrow{\vee}_\theta)(I(\theta), 0, w_0) & \textit{case 3} \end{cases}$$

   Please notice, that due to the weighted disjunction the weighted union is not associative. The semantics of $E = (E_1 \overleftarrow{\cup}_\theta E_2)$ *is analogously defined.*

9. intersections:
   - intersection $E = (E_1 \cap E_2) : I^*((E_1 \cap E_2)) =$
     $\{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge u_0 = I(\wedge)(v_0, w_0) > 0\}$.
   - weighted intersection $E = (E_1 \overrightarrow{\cap}_\theta E_2) : I^*((E_1 \overrightarrow{\cap}_\theta E_2)) =$
     $\{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge u_0 > 0\} \cup \{(u_0, v_1, \ldots, v_k) | (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge \forall v_0.(v_0, v_1, \ldots, v_k) \notin I^*(E_1) \wedge u_0 > 0\}$ *where*

$$u_0 = \begin{cases} I(\overrightarrow{\wedge}_\theta)(I(\theta), v_0, w_0) & \textit{case 1} \\ I(\overrightarrow{\wedge}_\theta)(I(\theta), 0, w_0) & \textit{case 2} \end{cases}$$

   Please notice, that due to the weighted conjunction the weighted intersection is not associative. Furthermore notice, that the definition of the

oarieli@mta.ac.il

weighted intersection does not correspond to a set intersection. The reason is the observation, that a weighted conjunction with one zero-valued operand can produce a non-zero result. The semantics of $E = (E_1 \overleftarrow{\cap}_\theta E_2)$ *is analogously defined.*

10. difference $E = (E_1 \setminus E_2) : I^*((E_1 \setminus E_2)) = \{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge \forall w_0.(w_0, v_1, \ldots, v_k) \notin I^*(E_2) \wedge u_0 > 0\} \cup \{(u_0, v_1, \ldots, v_k) | (v_0, v_1, \ldots, v_k) \in I^*(E_1) \wedge (w_0, v_1, \ldots, v_k) \in I^*(E_2) \wedge u_0 > 0\}$ *where*

$$u_0 = \begin{cases} I(\wedge)(v_0, I(\neg)(0)) & \textit{case 1} \\ I(\wedge)(v_0, I(\neg)(w_0)) & \textit{case 2} \end{cases}$$

11. cartesian product $E = E_a \times E_b$*:* $I^*(E_a \times E_b) = \{(u_0, v_1, \ldots, v_k, w_1, \ldots, w_l | (v_0, v_1, \ldots, v_k) \in I^*(E_a) \wedge (w_0, w_1, \ldots, w_l) \in I^*(E_b) \wedge u_0 = I(\wedge)(v_0, w_0)\}$
12. join $E = E_a \bowtie_{\#_{a_1}=\#_{b_1},\ldots,\#_{a_n}=\#_{b_n}} E_b$*:* $I^*(E_a \bowtie_{\#_{a_1}=\#_{b_1},\ldots,\#_{a_n}=\#_{b_n}} E_b) = \{(u_0, v_1, \ldots, v_k, w_1, \ldots, w_l | (v_0, v_1, \ldots, v_k) \in I^*(E_a) \wedge (w_0, w_1, \ldots, w_l) \in I^*(E_b) \wedge \forall i \in \{1, \ldots, n\}.v_{a_i} = w_{b_i} \wedge u_0 = I(\wedge)(v_0, w_0)\}$
    A join with an empty condition produces the cartesian product.

Please note, that every operation yields tuples with nonzero membership values.

6 Reducing $\mathcal{SDC}$ to $\mathcal{SA}$

In this section we show that the similarity algebra $\mathcal{SA}$ is as expressive as the similarity domain calculus $\mathcal{SDC}$.

Theorem 6.1. *Let* $\mathcal{SDC} = (\boldsymbol{U}, \boldsymbol{X}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *be a similarity relational domain calculus and* $\mathcal{SA} = (\boldsymbol{U}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *be a similarity relational algebra. For any* $\mathcal{SDC}$ *formula* F *there is an* $\mathcal{SA}$ *expression* E *that their queries are equivalent for any corresponding semantics:*

$$\{V(X_1), \ldots, V(X_n) | I_V^*(F(X_1, \ldots, X_n)) > 0\} = \{(v_1, \ldots, v_n) | (v_0, v_1, \ldots, v_n) \in I^*(E) \wedge v_0 > 0\}.$$

Proof. We prove Theorem 6.1 by constructively defining a *mapping* φ of an $\mathcal{SDC}$ formula F to a similarity algebra expression $E = \varphi(F)$:

1. case $F = R(Y_1, Y_2, \ldots, Y_m) : \varphi(F) = \pi_{\#_{v_1},\ldots,\#_{v_n}}(\sigma_{cond_c, cond_v}(R))$ where
   - $cond_c := \#_{i_1} = C_{i_1}, \ldots, \#_{i_l} = C_{i_l}$ where $Y_{i_1}, \ldots, Y_{i_l}$ are all constant names $C_{i_1}, \ldots, C_{i_l}$,
   - $cond_v := \#_{j_1} = \#_{k_1}, \ldots, \#_{j_o} = \#_{k_o}$ where $Y_{j_1} = Y_{k_1}, \ldots, Y_{j_o} = Y_{k_o}$ are all pairs of equally named variables reduced by reflexivity, symmetry, and transitivity on the attribute positions, and
   - $\#_{v_1}, \ldots, \#_{v_n}$ lists uniquely the attributes for all variables Y_{v_i} in the order of the variables.
2. case $F = Y_1 \delta Y_2$:

oarieli@mta.ac.il

- if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 \neq Y_2$ then $\varphi(F) = \sigma_{\#_1 \delta \#_2}(Dom_D \times Dom_D)$;
- if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 = Y_2$ then $\varphi(F) = \pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(Dom_D)))$;
- if $Y_1 \in \mathbf{X}$ and Y_2 is a constant $C \in \mathbf{C}$ then $\varphi(F) = \sigma_{\#_1 \delta C}(Dom_D)$;
- if $Y_2 \in \mathbf{X}$ and Y_1 is a constant $C \in \mathbf{C}$ then $\varphi(F) = \sigma_{C \delta \#_1}(Dom_D)$;
- if Y_1 and Y_2 are constants $C_1, C_2 \in \mathbf{C}$, respectively, then $\varphi(F) = \sigma_{C_1 \delta C_2}(\mathbf{1})$;

where $D = Dom(\delta)$.

3. case $F = (F_a(X_{a1}, \ldots, X_{ak}) \wedge F_b(X_{b1}, \ldots, X_{bl})) :$
$\varphi(F) = \pi_{\#_{p_1}, \ldots, \#_{p_n}}(\varphi(F_a) \bowtie_{cond_c} \varphi(F_b))$, where
   - $cond_c := \#_{v_1} = \#_{w_1}, \ldots, \#_{v_m} = \#_{w_m}$ where $X_{av_i} = X_{bw_i}$ for $i = 1, \ldots, m$ and
   - $\#_{p_1}, \ldots, \#_{p_n}$ lists uniquely the corresponding attributes for all variables $X_{a1}, \ldots, X_{ak}, X_{b1}, \ldots, X_{bl}$ in the order of the variables.
4. weighted `and`-cases:
case $F = (F_a(X_{a1}, \ldots, X_{ak}) \overrightarrow{\wedge}_\theta F_b(X_{b1}, \ldots, X_{bl})) : \varphi(F) = E_a \overrightarrow{\cap}_\theta E_b$, where

$$E_a = \pi_{\#_{i_1}, \ldots, \#_{i_m}}(\varphi(F_a) \times Dom_{D_{b_{o_1}}} \times \ldots \times Dom_{D_{b_{o_{z-k}}}}))$$
$$E_b = \pi_{\#_{j_1}, \ldots, \#_{j_m}}(\varphi(F_b) \times Dom_{D_{a_{p_1}}} \times \ldots \times Dom_{D_{a_{p_{z-l}}}}))$$

and the following conditions hold:
   - $\{X_{a1}, \ldots, X_{ak}, X_{b1}, \ldots, X_{bl}\}$ correspond bijectively to $att(E_a)$ and $att(E_b)$, respectively,
   - the attributes of E_a and of E_b occur in the order of the variables,
   - $z = |\{X_{a1}, \ldots, X_{ak}, X_{b1}, \ldots, X_{bl}\}|$, and
   - $D_{b_{o_i}} = Dom(X_{b_{o_i}})$ and $D_{a_{p_i}} = Dom(X_{a_{p_i}})$.

   The case for the left-oriented conjunction $\overleftarrow{\wedge}_\theta$ is defined analogously.
5. `or`-cases (unweighted and weighted): These cases are defined analogously to the weighted `and`-cases. $\vee$ is mapped to $\cup$, $\overleftarrow{\vee}_\theta$ is mapped to $\overleftarrow{\cup}_\theta$, and $\overrightarrow{\vee}_\theta$ is mapped to $\overrightarrow{\cup}_\theta$.
6. case $F = (\neg F_1(X_1, \ldots, X_m)) : \varphi(F) = (Dom_{D_1} \times \ldots \times Dom_{D_m}) \setminus \varphi(F_1)$, where $D_i = Dom(X_i)$. If F_1 has no variables ($m = 0$) then $\varphi(F) = \mathbf{1} \setminus \varphi(F_1)$.
7. case $F = (\exists X F_1(X_1, \ldots, X_{l-1}, X, X_{l+1}, \ldots, X_m)) :$
$\varphi(F) = \pi_{\#_{p_1}, \ldots, \#_{p_{m-1}}}(\varphi(F_1(X_1, \ldots, X_{l-1}, X, X_{l+1}, \ldots, X_m)))$, where
$\#_{p_1}, \ldots, \#_{p_{m-1}}$ lists the corresponding attributes for all variables $X_1, \ldots, X_{l-1}, X_{l+1}, \ldots, X_m$ in the order of the variables.
8. case $F = (\exists_k X F_1(X_1, \ldots, X_{l-1}, X, X_{l+1}, \ldots, X_m)) :$
$\varphi(F) = \pi^k_{\#_{p_1}, \ldots, \#_{p_{m-1}}}(\varphi(F_1(X_1, \ldots, X_{l-1}, X, X_{l+1}, \ldots, X_m)))$, where
$\#_{p_1}, \ldots, \#_{p_{m-1}}$ lists the corresponding attributes for all variables $X_1, \ldots, X_{l-1}, X_{l+1}, \ldots, X_m$ in the order of the variables.
9. case $F = (\forall X F_1) : \varphi(F) = \varphi((\neg(\exists X(\neg F_1))))$.
10. case $F = (\forall_k X F_1) : \varphi(F) = \varphi((\neg(\exists_k X(\neg F_1))))$.

oarieli@mta.ac.il

Mapping an $\mathcal{SDC}$-formula to the similarity algebra can produce similarity algebra expressions containing the Dom-operation. In such cases, the expression is domain dependent. If, however, an $\mathcal{SDC}$-formula is evaluable (see [9]) and we map it to the similarity algebra then we can replace all occurring Dom-operations by similarity algebra expressions over database relations and obtain domain independent algebra expressions.

Theorem 6.2. *Let* $\mathcal{SDC} = (\boldsymbol{U}, \boldsymbol{X}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *be a similarity relational domain calculus and* $\mathcal{SA} = (\boldsymbol{U}, \Delta, \boldsymbol{C}, \boldsymbol{D}, Dom, \boldsymbol{R}, \Theta)$ *be a similarity relational algebra. For any* evaluable $\mathcal{SDC}$ *formula* F *there is a* domain independent $\mathcal{SA}$ *expression* E *that their queries are equivalent for any corresponding semantics:*

$$\{V(X_1), \ldots, V(X_n) | I_V^*(F(X_1, \ldots, X_n)) > 0\} = \{(v_1, \ldots, v_n) | (v_0, v_1, \ldots, v_n) \in I^*(E) \wedge v_0 > 0\}.$$

Proof. We prove Theorem 6.2 by modifying the *mapping* φ from the proof of theorem 6.1 to the mapping φ^* of an evaluable $\mathcal{SDC}$ formula F to a similarity algebra expression $\varphi^*(F)$.

Let $E = \{X_1, X_2, \ldots, X_n | F(X_1, X_2, \ldots, X_n)\}$ be an evaluable $\mathcal{SDC}$ query and let

$$rel(X) = ((E_1) \cap \ldots \cap (E_m)) \text{ as defined in [9].}$$

1. For every atom $Y_1 \delta Y_2$ within F replace the corresponding algebra term
   a) $\sigma_{\#_1 \delta \#_2}(Dom_D \times Dom_D)$ in $\varphi(F)$ by $\sigma_{\#_1 \delta \#_2}(rel(Y_1) \times rel(Y_2))$ if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 \neq Y_2$,
   b) $\pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(Dom_D)))$ in $\varphi(F)$ by $\pi_{\#_1}(\sigma_{\#_1 \delta \#_2}(\pi_{\#_1, \#_2}(rel(Y_1))))$ if $Y_1, Y_2 \in \mathbf{X} \wedge Y_1 = Y_2$,
   c) $\sigma_{\#_1 \delta C}(Dom_D)$ in $\varphi(F)$ by $\sigma_{\#_1 \delta Y_2}(rel(Y_1))$ if $Y_1 \in \mathbf{X}, Y_2 \in \mathbf{C}$, and
   d) $\sigma_{C \delta \#_1}(Dom_D)$ in $\varphi(F)$ by $\sigma_{Y_1 \delta \#_1}(rel(Y_2))$ if $Y_1 \in \mathbf{C}, Y_2 \in \mathbf{X}$.
2. weighted conjunctions and disjunctions: For every weighted conjunction $(F_a(X_{a1}, \ldots, X_{ak}) \overset{\rightarrow}{\wedge}_\theta F_b(X_{b1}, \ldots, X_{bl}))$ in F replace the corresponding algebra term $E_a \overset{\rightarrow}{\cap}_\theta E_b$ in $\varphi(F)$ by $E'_a \overset{\rightarrow}{\cap}_\theta E'_b$ where

$$E'_a = (\pi_{\#_{i_1}, \ldots, \#_{i_m}}(\varphi(F_a) \times rel'(X_{b_{o_1}}) \times \ldots \times rel'(X_{b_{o_{z-k}}})))$$
$$E'_b = (\pi_{\#_{j_1}, \ldots, \#_{j_m}}(\varphi(F_b) \times rel'(X_{a_{p_1}}) \times \ldots \times rel'(X_{a_{p_{z-l}}})))$$

   and the following conditions hold:
   – $\{X_{a1}, \ldots, X_{ak}, X_{b1}, \ldots, X_{bl}\}$ correspond bijectively to $att(E'_a)$ and $att(E'_b)$, respectively,
   – the attributes of E'_a and of E'_b occur in the same order, and
   – $z = |\{X_{a1}, \ldots, X_{ak}, X_{b1}, \ldots, X_{bl}\}|$.
   –
$$rel'(X) = \begin{cases} rel(X)) \cup \mathbf{0} \text{ if } X \text{ is bound to } \exists \text{ or } \forall \\ rel(X) \qquad \text{otherwise} \end{cases}$$

   These cases for $\overset{\leftarrow}{\wedge}_\theta, \vee, \overset{\rightarrow}{\vee}_\theta$, and $\overset{\leftarrow}{\vee}_\theta$ are defined analogously.

oarieli@mta.ac.il

3. For every negation $(\neg F_1(X_1, \ldots, X_m))$ within F replace the corresponding algebra term
$(Dom_{D_1} \times \ldots \times Dom_{D_m}) \setminus \varphi(F_1)$ in $\varphi(F)$ by $(rel(X_1) \times \ldots \times rel(X_m)) \setminus \varphi(F_1)$.

Now we are able to formulate weighted similarity queries in declarative manner using $\mathcal{SDC}$ and to reduce them to an equivalent algebra expression. Thus, the evaluation of those similarity queries can be processed based on the proposed SA.

7 The Phantom Problem

Assume the following evaluable calculus formula $\exists X(R_1(X) \vee R_2(C_1))$ is given and the relation $I(R_1)$ is an empty relation. If additionally the second relation contains the constant value, then the formula is satisfied and returns true.

Consider now the mapping to the similarity algebra. Following the mapping rule for the $\vee$-construct we have to unite ($\cup$) two algebra expressions. Since both expressions are not union compatible the right expression needs to be combined with the relation $I(R_1)$ applying the cartesian product. However, in our case the relation $I(R_1)$ is empty and the cartesian product with an empty relation produces always an empty relation. Therefore, the complete algebra expression returns no tuple. This contradicts the result of the calculus expression and we call it the *phantom problem*.

We solve this problem by never using an empty relation for becoming union compatible. This is achieved by applying the '$\cup$ $\mathbf{0}$' operation to R_1[7].

The phantom problem becomes worse if we replace $\exists X$ by $\exists_k X$: $(\exists_k X(R_1(X) \vee R_2(C_1)))$. In this case, the formula is not domain independent anymore. The reason is that the result depends on the number of elements of the domain for X. Therefore, for evaluable formulas we do not allow any disjunction or weighted conjunction with a subformula independent from X below an $\exists_k X$ or a $\forall_k X$ construct.

8 Example

In order to demonstrate the potential of our approach we will demonstrate the transformation process on an example using a fabric seller database. Information of different fabrics is stored in Table *Fabric*. To each fabric there exist several images that are stored in Table *Image*. A small fragment of these tables is shown below. For similarity calculation between the given images and those in the database different similarity operators are available, e. g. $\sim_C$ to determine the similarity regarding the color feature.

[7] See rule 2 in the proof for Theorem 6.2.

oarieli@mta.ac.il

Fabric

FId	Name	Quality
F001	tartan_0815	high
F002	tartan_0816	low
F003	blue_stripes	high
...	...	...

Image

IId	FId	Image
I001	F001	tartan_0815_1.jpg
I002	F001	tartan_0815_2.jpg
I003	F002	tartan_0816_1.jpg
I004	F003	blue_stripes_1.jpg
...	...	...

Consider the following query:

> »*Retrieve name and quality of those fabrics that have a high quality or that match the given query image (C_{image}) in color and texture. Thereby, color is twice as important as texture and the quality criteria is three times less important than the similarity to the given image.*«

Formulating queries in an declarative way is much more comfortable to the user since the user only describes what he is interested in and not how it can be obtain from the database. Therefore, we can directly formulate this query as $\mathcal{SDC}$-expression, which is illustrated in Fig. 1. Compared to the algebra expression in Fig. 2 the calculus expression is less complex and easier to understand.

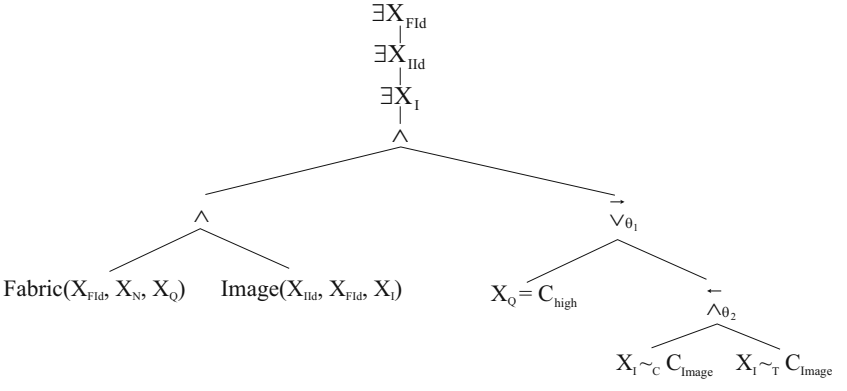


Fig. 1. Query tree of the $\mathcal{SDC}$-expression with the free variables X_N and X_Q.

Of course, querying directly in calculus is still too difficult for a non-expert user. For that reason we are currently designing a graphical QBE-like query language. From there, a mapping onto our calculus can be easily performed.

Mapping our $\mathcal{SDC}$ query expression yields a complex $\mathcal{SA}$ expression, which corresponds to the query tree shown in Figure 2.

We use Fagin's weighting formula [10] for the weighted combination of similarity values. As underlying scoring function we employ the functions *min* for conjunction and *max* for disjunction. In order to work appropriate with the

oarieli@mta.ac.il

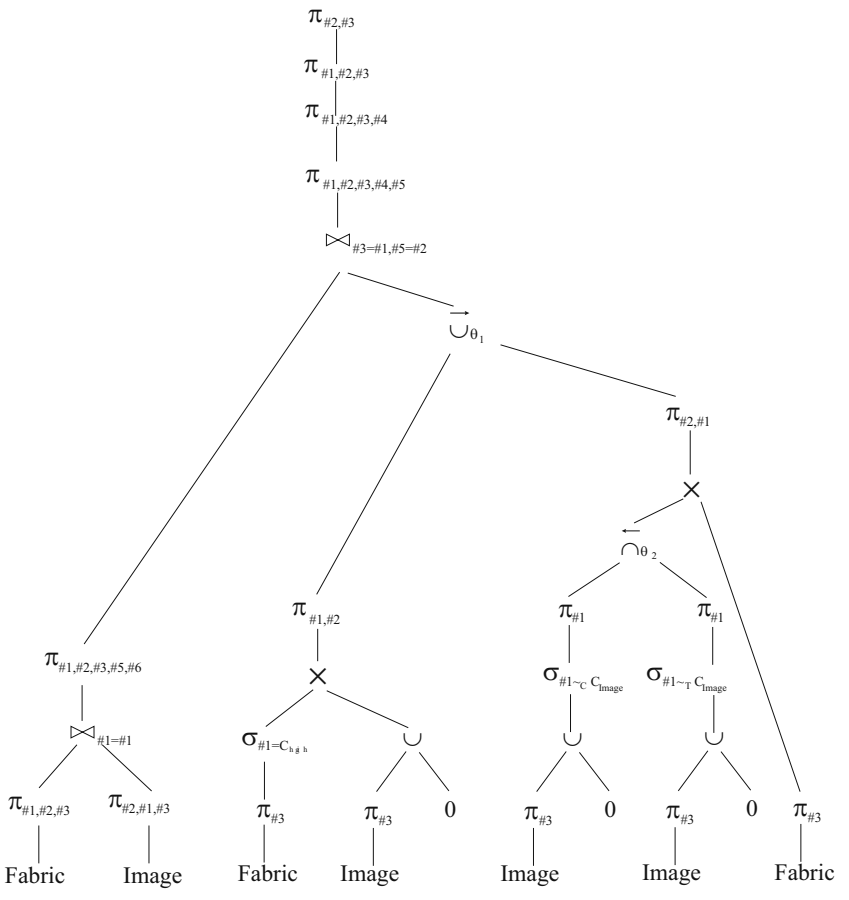


Fig. 2. Query tree of the generated $\mathcal{SA}$-expression.

weighting formula we transform our operator weights to operand weights as shown in Section 3.

Due to the specified user preferences, which state that the quality criteria is three times less important than the similarity criteria we obtain for $\overrightarrow{\triangledown}_{\theta_1}$ the weight $\theta_1 = 1/2$. The corresponding operand weights are therefore $\theta_{quality} = 1/4$ and $\theta_{\sim} = 3/4$. Since color should be twice as important as texture, the weight θ_2 for the operation $\overleftarrow{\wedge}_{\theta_2}$ is $1/3$, which corresponds to the operand weights $\theta_{\sim_C} = 2/3$ and $\theta_{\sim_T} = 1/3$. The constant C_{high} is evaluated to the string '`high`' and the constant C_{Image} corresponds to a query image. The calculated similarity values for the subformulas are summarized in Table *Similarity Values*. The query result is shown in Table *Result*. The final projection eliminates duplicates. Thus, the tuple with the highest score of the duplicates is taken. For fabric *tartan_0815*

oarieli@mta.ac.il

the tuple with score 0.9 is chosen, whereas the duplicate tuple with score 0.69 is omitted.

Similarity Values

IId	$\sim_C$	$\sim_T$	$\overleftarrow{\wedge}_{\theta_2}$	$\overrightarrow{\vee}_{\theta_1}$
I001	0.7	0.2	0.37	0.69
I002	0.8	0.9	0.80	0.90
I003	0.9	0.4	0.57	0.57
I004	0.1	0.8	0.10	0.55
...	...	...	...	...

Result

Name	Quality	Score
tartan_0815	high	0.90
tartan_0816	low	0.57
blue_stripes	high	0.55
...	...	...

9 Conclusion and Future Work

Our approach defines a framework for transforming declarative similarity queries into an appropriate expression of an extended relational algebra. We enhanced traditional relational domain calculus by vagueness and the aspect of weighting. We left the specification of exact fuzzy-operations and weighted scoring functions unspecified, so that our approach can be adapted and flexible extended to meet the needs of various scenarios, e. g. image retrieval or other multimedia applications.

So far, our proposed language comprises core functionality only. We provide adequate operators for dealing with imprecision and weights. Further, we proposed the parameterizable quantifiers $\exists_k$ and $\forall_k$. In future, we will expand our language by new constructs. We plan to add a similarity join [24,16] as wells as *top*- and *cut*-operations [17]. In addition, we intend to support aggregations and other functions on result variables. Another aspect we want to consider is to weight the operators itself, that is, to modify their behavior by a parameter, e. g. to soften a conjunction in direction to the disjunction. Since formulating queries in a calculus is not very user friendly, we aim to design a graphical query language in a QBE-like fashion. Then, a mapping onto our similarity calculus can be easily performed.

Reducing calculus expressions to algebra produces often very complex expressions requiring subsequent algebraic optimization. We are planning to develop appropriate optimization strategies. Therefore, special optimization rules adapted to our mapping rules need to be developed. In addition to a subsequent optimization we are working on implicit optimization during the mapping. Furthermore, we aim to extend the relational database model in order to explicitly store and query imprecise data.

References

1. Zadeh, L.A.: Fuzzy Logic. IEEE Computer **21** (1988) 83–93
2. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM **13** (1970) 377–387
3. Codd, E.F.: A Database Sublanguage Founded on the Relational Calculus. In: ACM SIGFIDET Workshop on Data Description, Access and Control. (1971) 35–61

oarieli@mta.ac.il

4. Codd, E.F.: Relational Completeness of Data Base Sublanguages. In Rustin, R., ed.: Data Base Systems. Volume 6. Prentice Hall, Englewood Cliffs, NJ (1972) 65–98
5. Codd, E.F.: Relational Database: A Practical Foundation for Productivity. Communications of the ACM **25** (1982) 109–117
6. Maier, D.: The Theory of Relational Databases. Computer Science Press, Rockville, MD (1983)
7. Ullman, J.D.: Principles of Database Systems. Computer Science Press (1982)
8. Biskup, J.: Grundlagen von Informationssystemen. Vieweg-Verlag, Braunschweig, Wiesbaden (1995)
9. Schmitt, I., Schulz, N.: Safe Reduction of Similarity Calculus to Similarity Algebra. Preprint, Fakultät für Informatik, Universität Magdeburg (to appear)
10. Fagin, R., Wimmers, E.L.: A Formula for Incorporating Weights into Scoring Rules. Special Issue of Theoretical Computer Science (2000)
11. Schulz, N., Schmitt, I.: Relevanzwichtung in komplexen Ähnlichkeitsanfragen. In Weikum, G., Schöning, H., Rahm, E., eds.: Datenbanksysteme in Business, Technologie und Web, BTW'03, 10. GI-Fachtagung, Leipzig, Februar 2003. Lecture Notes in Informatics (LNI) Volume P-26, Bonn, Gesellschaft für Informatik (2003) 187–196
12. Kießling, W.: Foundations of preferences in database systems. In: Proceedings of the 28th International Conference on Very Large Databases (VLDB), Hong Kong, China. (2002) 311–322
13. Bolloju, N.: A Calculus for Fuzzy Queries on Fuzzy Entity-Relationship Model. Technical Report 94/26, Department of Information Systems at the City Polytechnic of Hong Kong (1994)
14. Galindo, J., Medina, J.M., Pons, O., Cubero, J.C.: A Server for Fuzzy SQL Queries. In Andreasen, T., Christiansen, H., Larsen, H.L., eds.: Flexible Query Answering Systems, Third International Conference, FQAS'98, Roskilde, Denmark, May 13-15, 1998. Volume 1495 of Lecture Notes in Computer Science., Springer (1998) 164–174
15. Takahashi, Y.: Fuzzy Database Query Languages and Their Relational Completeness Theorem. IEEE Transaction on Knowledge and Data Engineering **5** (1993) 122–125
16. Adali, S., Bonatti, B., Sapino, M.L., Subrahmanian, V.S.: A Multi-Similarity Algebra. In: ACM SIGMOD Int. Conf. on Management of Data, Seattle, Washington, USA (1998) 402–413
17. Ciaccia, P., Montesi, D., Penzo, W., Trombetta, A.: Imprecision and user preferences in multimedia queries: A generic algebraic approach. In Schewe, K.D., Thalheim, B., eds.: FoIKS: Foundations of Information and Knowledge Systems, First International Symposium, FoIKS 2000, Burg, Germany, February 14-17, 2000. Volume 1762 of Lecture Notes in Computer Science., Springer (2000) 50–71
18. Fuhr, N., Rölleke, T.: A Probabilistic Relational Algebra for the Integration of Information Retrieval and Databases Systems. ACM Transactions on Information Systems (TOIS) **15** (1997) 32–66
19. Sung, S.Y.: A Linear Transform Scheme for Combining Weights into Scores. Technical report, Rice University (1998)
20. Schulz, N., Schmitt, I.: A Survey of Weighted Scoring Rules in Multimedia Database Systems. Preprint 7, Fakultät für Informatik, Universität Magdeburg (2002)

oarieli@mta.ac.il

21. Carson, C., Belongie, S., Greenspan, H., Malik, J.: Region-based image querying. In: Proc. of the IEEE Workshop CVPR '97 Workshop on Content-Based Access of Image and Video Libraries, Puerto Rico. (1997) 42–49
22. Ortega, M., Rui, Y., Chakrabarti, K., Porkaew, K., Mehrotra, S., Huang, T.S.: Supporting Ranked Boolean Similarity Queries in MARS. tkde **10** (1998) 905–925
23. Gelder, A.V., Topor, R.W.: Safety and Translation of Relational Calculus Queries. ACM Transactions on Database Systems **16** (1991) 235–278
24. Atnafu, S., Brunie, L., Kosch, H.: Similarity-Based Operators and Query Optimization for Multimedia Database Systems. In Adiba, M.E., Collet, C., Desai, B.C., eds.: International Database Engineering & Applications Symposium, IDEAS '01, July 16-18, 2001, Grenoble, France, Proceedings, IEEE Computer Society (2001) 346–355

oarieli@mta.ac.il

Challenges in Fixpoint Computation with Multisets*

Nematollaah Shiri and Zhi Hong Zheng

Concordia University
Dept. of Computer Science
1400 Maisonneuve Blvd. West
Montreal, Quebec, Canada H3G 1M8
{shiri,zh_zheng}@cs.concordia.ca

Abstract. Uncertainty management has been a challenging issue in AI and database research. Logic database programming with its declarative advantage and its top-down and bottom-up query processing techniques has been an attractive formalism for representing and manipulating uncertain information, and numerous frameworks with uncertainty has been proposed. These proposals address fundamental issues of modeling, semantics, query processing and optimization, however, one important issue which remains unaddressed is efficient implementation of such frameworks. In this paper, we illustrate that the standard seminaive evaluation method does not have a counterpart in general in these frameworks. We then propose a desired semi-naive algorithm, which extends the corresponding standard method, and establish its equivalence with the naive method with uncertainty. We implemented the algorithm and conducted numerous tests. Our experimental results indicate that the proposed technique is practical and supports efficient fixpoint computation with uncertainty. We believe that the method is also useful in a more general context of fixpoint computation with aggregations.

1 Introduction

Many real-life applications require an ability to represent and reason with uncertain information. Examples include diagnostic applications, data mining, scientific databases, and pattern and image databases. Answering *complex queries* against such applications requires that certainties associated with answers to simple queries be *combined* using some *well-grounded principles* and in a *meaningful way*. Uncertainty is a form of imperfection in information, which arises when the truth of the information is not established definitely. More precisely, uncertainty is the "degree" of truth of information pieces as estimated by an individual or a sensor device, which may be represented by associating with the information, a value coming from an appropriate domain.

* This research was supported in part by grants from the National Sciences and Engineering Research Council of Canada (NSERC).

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 273–290, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

Uncertainty management has been a challenging issue in AI and database systems for a long time. Parsons [18] provides a survey of works on the more general subject of imperfect information in AI and databases. Numerous frameworks have been proposed for uncertainty by extending the standard logic database programming with its advantages of modularity and its powerful top-down and bottom-up query processing techniques. The proposed frameworks typically combine *deduction with some formalism for uncertainty*, and as in the standard case, they offer a declarative semantics of programs. For practical reasons, a desired such framework should admit efficient implementation and computation. On the operational side, this is supported by a sound and complete (or weakly complete, in some cases) proof procedure and a corresponding fixpoint semantics.

There are a number of basis on which these frameworks may differ. On the basis of their underlying mathematical foundation of uncertainty, these frameworks may vary and include probability theory [7,8,16,17], fuzzy set theory [21, 24], multi-valued logic [3,4,5,6], possibilistic logic [2], evidence theory [15], and hybrid of numerical and non-numerical formalisms [7,9]. On the basis on which uncertainties are associated with the facts and rules in a program, we classified [10] the approaches of these frameworks into two: *annotation based* (AB, for short) [23,5,15,16,17,6] and *implication based* (IB) [24,3,4,2,8,9]. Our earlier work [12], includes a comprehensive comparison of these approaches. For a survey of research on uncertainty in logic programming and deductive databases, interested reader is referred to [11].

A typical rule r in an AB framework is an expression of the form:

$$H : f(\beta_1, \ldots, \beta_k) \longleftarrow B_1 : \beta_1, \cdots, B_n : \beta_n.$$

where H and B_i's are atoms, β_i is an annotation constant or variable, and f is a function to compute the certainty of the rule head by "combining" the certainties of the subgoal in the rule body. Alternate derivations of the same atom from the program are "combined", using a user-defined disjunction function.

By contrast, a rule r in an IB framework is an expression of the form:

$$H \stackrel{\alpha}{\longleftarrow} B_1, \cdots, B_n.$$

where H and B_i's are atoms, and α is a certainty value. This rule asserts that: *the certainty that the rule body implies the head is* α. Aside from the syntactic difference between the AB and IB approaches, there are important differences in that in principle annotation functions in AB frameworks are unconstrained, or at least not discussed, whereas the computation in IB frameworks is typically constrained by some principles making sure the certainty computation makes intuitive sense.

To introduce some basic concepts we need in our work, let us consider van Emden's framework [24], the first language proposed for logic programs with uncertainty. As in the standard case, a rule r in [24] is applicable when each subgoal B_i in the rule body is true (to some extent). When r is applied, it yields a ground instance A of the rule head H. Besides, we also need to consider the presence of certainty values and functions in rule applications. To explain this, suppose I is an interpretation, which basically assigns to each ground atom B, a truth value $I(B)$ in the closed unit interval $[0, 1]$. Therefore, in the above derivation of A, the

oarieli@mta.ac.il

certainty σ assigned to A according to [24] would be $\alpha \times min\{I(B_1), \cdots, I(B_n)\}$. Here, the certainty of the rule body is determined by taking the *minimum* of the certainties associated with the instances of the subgoals in the rule body which contributed to this derivation of A. We may derive A multiple times from this rule and/or other rules in the program. We thus need to "combine" these alternate derivations of A. In [24], this is done by taking the *maximum* of all certainties derived for A. Note the use of three different "combination" functions involved in this process: (i) *min* was used as the *conjunction* function to define the certainty of the rule body as a whole, (ii) product ($\times$) was used as the *propagation* function to define the certainty associated with the atom derived by this rule application, and (iii) *max* was used as the *disjunction* function to combine alternate derivations of the same atom A into a single certainty of A. This iterative process continues until it reaches an iteration at which no atom is derived with a higher/better certainty, assuming initially every atom is assigned the least certainty value 0, which corresponds to *false* in the standard logic.

Efficient query processing and optimization for standard logic programs have been studied extensively and numerous compile-time and run-time techniques have been proposed and implemented in existing inference systems. Ceri et al. [1] includes an excellent survey of research in this direction.

In the context of logic frameworks with uncertainty, even though there have been numerous proposals for AB/IB frameworks, there has been little progress in their effective and efficient implementation. Leach and Lu [13] discuss implementation issues in the context of a multi-valued AB framework with a set-based semantics. However, query processing could be complicated when the semantics is based on multisets. In our attempt to redress this situation, we have been trying to lift the rich body of theory and techniques developed in the standard framework to these extended frameworks. Rather than attempt this for individual frameworks, our approach has been to address this problem in a "framework independent" manner. To this end, we proposed a generic IB framework, called the *parametric framework* [10], which unifies and/or generalizes all the IB frameworks, and established that query programs in this framework have an equivalent declarative, fixpoint, and proof theoretic semantics. The parametric framework also provided a basis to study query optimization for the IB frameworks with uncertainty [12].

An indispensable, efficient run-time optimization technique for standard logic programs and deductive databases proposed as an alternative to the *naive* method is the *semi-naive* method, which tries to avoid or minimize repeated applications of rules at every iteration step. This is basically done by limiting the rule applications, at every iteration, to those rules for which we derived, in the previous iteration, at least "one new atom" in the rule body. It is also desired to use the semi-naive method for efficient evaluation of programs in the parametric framework.

In this paper, we study efficient evaluation of logic programs with uncertainty. As we illustrate in the following section, a "straight" extension of the standard semi-naive method to take into account the presence of certainty values will not

oarieli@mta.ac.il

work in general, simply because the results may not always coincide with the results obtained by the corresponding naive method with uncertainty. Our contribution in this work is thus proposing a "careful" extension of the standard semi-naive method for efficient evaluation of logic programs with uncertainty and establish its equivalence with the corresponding naive method with uncertainty. The ideas employed in this proposal may be adopted and used for the AB frameworks as well. We believe the proposed solution could be adopted in a more general context of fixpoint computation with aggregation.

The rest of this paper is organized as follows. Next, we present a motivating example. Section 3 includes a quick review of the parametric framework as a background. Section 4 provides a classification of disjunctions functions, which is essential to this wok. In section 5, we propose a semi-naive algorithm for fixpoint computation with uncertainty and establish its equivalence with the corresponding naive method. Section 6 includes highlights of a prototype system we developed which implements the proposed method. We also present our experimental results indicating the efficiency of the system for evaluating programs with uncertainty. In section 7, we discuss conditions under which existing inference engines may be used for evaluating programs with uncertainty. Finally we provide a summary and concluding remarks. We assume that the reader is familiar with the foundations of logic programs [14] and deductive databases [1].

2 A Motivating Example

We now illustrate that the naive and semi-naive methods in the standard framework may produce different results when uncertainty is present. For this, it is enough to limit the analysis to the following program P_1 with uncertainty expressed in propositional logic.

Example 2.1. Let P_1 be the following program with uncertainty.

$$r_1 : B \xleftarrow{0.5} .$$
$$r_2 : C \xleftarrow{0.8} .$$
$$r_3 : A \xleftarrow{1} C; \qquad \langle ind, \times, \_ \rangle.$$
$$r_4 : A \xleftarrow{0.6} B,\ A; \quad \langle ind, \times, \times \rangle.$$

The underlying certainty lattice in P_1 is $\langle [0,1], \leq, min, max \rangle$, with min as the meet operator and max as the join. The two facts r_1 and r_2 define the atom-certainty pairs $B : 0.5$ and $C : 0.8$, respectively. The certainty associated with rule r_3 is 1 and that of rule r_4 is 0.6. The triple $\langle f_d, f_p, f_c \rangle$ associated with each rule indicates the disjunction, propagation, and conjunction functions, respectively. The disjunction function ind in r_4 is defined as $ind(\alpha, \beta) = \alpha + \beta - \alpha \times \beta$. It should be clear that the particular choice of the conjunction function in r_3 is immaterial, indicated as "_", in the sense that any "reasonable" conjunction function would return 0.8 as the certainty of the body of r_3.

Let us first consider a fixpoint naive evaluation of P_1, and then "adapt" a semi-naive method for evaluating it. We use $A : \alpha$ to denote an atom A and

oarieli@mta.ac.il

its associated certainty α, and use I_j to denote the collection of atom-certainty pairs obtained at iteration j. Initially, $I_0 = \{A : 0,\ B : 0,\ C : 0\}$, that is every atom is false. At iteration 1, we get $I_1 = \{B : 0.5,\ C : 0.8\}$ from r_1 and r_2. These two facts are derived at every following iteration. This, as in the standard case, is a source of inefficiency. At iteration 2, we can apply r_3 and get $I_2 = \{A : 0.8,\ B : 0.5,\ C : 0.8\}$. At iteration 3, we have two derivations of A; one from r_4 with certainty $0.6 \times 0.5 \times 0.8 = 0.24$, and the other from r_3 with certainty 0.8. The combined certainty of A is thus $ind(0.8, 0.24) = 0.848$, and hence $I_3 = \{A : 0.848,\ B : 0.5,\ C : 0.8\}$. We continue to get two derivations of A at every following iteration. At iteration 4, we get $A : 0.8$ from r_3 and $A : 0.2544$ from r_4, which when combined give $A : 0.85088$, and thus $I_4 = \{A : 0.85088,\ B : 0.5,\ C : 0.8\}$. Since the certainty of A increases at every iteration, this process goes on and terminates only at the limit.[1]

To obtain the evaluation result at the limit, we adopt a recurrence relation based technique from [5], as follows. Let α_n denote the certainty of A at iteration n. Then, the certainty α_{n+1} of A at iteration $n+1$ would be expressed as the recurrence relation: $\alpha_{n+1} = 0.8 + 0.6 \times 0.5 \times \alpha_n - 0.8 \times (0.6 \times 0.5 \times \alpha_n)$. Let α denote the certainty of A at the limit, i.e., when n approaches ω. This is obtained when $\alpha_{n+1} = \alpha_n$, indicating no change in A's certainty. Using this equality, the above recurrence relation may be reduced to $\alpha = 0.8 + 0.3\alpha - 0.24\alpha$, solving which yields $\alpha = 0.8/0.94$. Therefore, the least fixpoint semantics of P_1 using the naive method would be $I_\omega = \{A : 0.85106,\ B : 0.5,\ C : 0.8\}$.

Next we show that a "straight" extension of the semi-naive method will not work for evaluating P_1. This would also suggest that we may not use existing inference systems, as will be explained shortly. This is an unfortunate In fact, we noticed the above undesirable behaviour when our attempts to implement the probabilistic logic framework of Lakshmanan and Sadri [8] in both XSB and CORAL failed. This suggests that, unfortunately, we may not be able to take advantage of existing powerful and efficient systems such as CORAL and XSB to evaluate some logic programs with uncertainty. Careful extensions are thus required in order to take advantage of these systems.

The basic idea of the standard semi-naive method is to apply at each iteration i, every rule r for which we derived "something new" for the rule body at iteration $i-1$. This something new could be a new atom for a subgoal in the rule body, as in the standard case, or an atom but with a "better" certainty in our context. Extending this idea, we derive $B : 0.5$ and $C : 0.8$ at iteration 1. At step 2, we only apply r_3 and derive $A : 0.8$. At iterations 3 and after, we only apply r_4 and continue to derive A with a better certainty at every following iteration. At iteration 3, we get $A : 0.24$ from r_4, which is then combined with the best certainty 0.8 of A known thus far, and hence $I_3 = \{A : 0.848,\ B : 0.5,\ C : 0.8\}$. At iteration 4, r_4 derives A with certainty 0.2544 and consequently $I_4 = \{A : 0.886688,\ B : 0.5,\ C : 0.8\}$, which is incorrect and continues to yield accumulative wrong result in the limit. The recurrence relation that corresponds to this evaluation is: $\alpha_{n+1} = \alpha_n + 0.6 \times 0.5 \times \alpha_n - 0.3 \times \alpha_n^2$, solving which

[1] The underlying fixpoint operator is shown to be monotone and continuous [10].

oarieli@mta.ac.il

yields $\alpha = 1$. That is, $I_\omega = \{A : 1,\ B : 0.5,\ C : 0.8\}$, which is incorrect, as predicted. This example suggests that care must be paid when extending the standard semi-naive method to develop a desired, efficient semi-naive method for programs with uncertainty.

Some explanation is in order. An important property of a disjunction function f_d (*ind* in P_1) which causes this undesirable behaviour is that its result is always larger than its arguments (unless at least one of them is 1, the top). Note that unlike *max* which is a disjunction often used in programs with uncertainty, *ind* is sensitive to duplicates in that $ind(\alpha, \alpha) = 2\alpha - \alpha^2 \neq \alpha$, and hence every single derivation of an atom counts, in general. We thus need to collect the derivations as a multiset. Note that this is not to suggest that a user in a framework with uncertainty is forced to conceive of uncertainty as multisets. However, because of the way the fixpoint evaluation proceeds, different derivations of the same atom may yield a certainty multiple times. In such a case, a set-based structure often assumed may not correctly capture the semantics. It is the use of disjunctions such as *ind* that complicates query processing with uncertainty, an issue that is dealt with in this paper.

This example also illustrates that the standard semi-naive method does not have a counterpart in deductive databases with uncertainty. It suggests that special book keeping is necessary, in general, in a semi-naive evaluation of programs with uncertainty to handle disjunctions such as *ind*. We remark that top-down inference systems compute in a similar way as explained, and hence similar book keeping maybe required also in such systems as well.

3 The Parametric Parameter: A Review

In this section, we recall the basic concepts and development of the *parametric framework* [10]. This includes an introduction of the parameters as well as the declarative and fixpoint semantics.

The idea of the parametric framework inspired from our observation that a user in an IB framework specifies in the program, implicitly or explicitly, the following notions, or *parameters* as we called, defined as follows. (1) A (finite or an infinite) set of certainty values, called the *certainty domain*, denoted by $\mathcal{T}$. It is often a partially ordered set and is assumed to be a complete lattice, (2) A family $\mathcal{F}_p$ of *propagation functions* each of which is a mapping from $\mathcal{T} \times \mathcal{T}$ to $\mathcal{T}$. (3) A family $\mathcal{F}_c$ of *conjunction functions*, modeled in general as a mapping from finite multisets over $\mathcal{T}$ to $\mathcal{T}$, since there could be several subgoals in a rule body. (4) A family $\mathcal{F}_d$ of *disjunction functions*, each of which is defined as a mapping from finite multisets over $\mathcal{T}$ to $\mathcal{T}$. Intuitively, a conjunction function returns the certainty of the rule body as a whole, and a propagation function associated with a rule combines the certainty of the rule body and the rule certainty and yields a certainty for the head atom. The disjunction function associated with a predicate is used to combine alternative derivations of the same ground atom into a single certainty of that atom. We refer to all these functions collectively

oarieli@mta.ac.il

as *combination functions* $\mathcal{F} = \mathcal{F}_c \cup \mathcal{F}_p \cup \mathcal{F}_d$. Using these parameters, we formally define the syntax of the parametric programs, as follows.

Definition 3.1. *A parametric program (p-program) P is a 5-tuple $\langle \mathcal{T}, \mathcal{R}, \mathcal{D}, \mathcal{P}, \mathcal{C} \rangle$, whose components are defined as follows:*

1. *$\mathcal{L} = <\mathcal{T}, \preceq, \otimes, \oplus>$ is a complete lattice, where $\mathcal{T}$ is a set of certainty values, partially ordered by $\preceq$, $\otimes$ is the meet operator, and $\oplus$ is the join. The least element of the lattice is denoted by $\bot$ and the greatest element by $\top$.*
2. *$\mathcal{R}$ is a finite set of parametric rules (p-rules, for short), each of which is an expression of the form:*
$$H \xleftarrow{\alpha} B_1, \cdots, B_n.$$
where H, $B_1, \cdots, B_n$ are atomic formulas, and α is a certainty value in $\mathcal{T} - \{\bot\}$.
3. *$\mathcal{D}$ is a mapping that associates with each predicate symbol p in P, a disjunction function in $\mathcal{F}_d$.*
4. *$\mathcal{P}$ is a mapping that associates with each p-rule in P a propagation function in $\mathcal{F}_p$.*
5. *$\mathcal{C}$ is a mapping that associates with each p-rule in P, a conjunction function in $\mathcal{F}_c$.*

For ease of presentation, we write a p-rule as follows:
$$r: \; H \xleftarrow{\alpha_r} B_1, \ldots, B_n; \quad \langle f_d, f_p, f_c \rangle.$$
where f_d is the disjunction function, f_p is the propagation function, and f_c is the conjunction function. If A is an atomic formula, we use $\pi(A)$ to denote the predicate symbol of A. For instance, if $A = p(X_1, \cdots, X_k)$, then $\pi(A) = p$. We also use $Disj(\pi(A))$ to denote the disjunction function associated with the predicate symbol of A. That is, if this disjunction function is f_d, then $Disj(\pi(A)) = f_d$.[2]

We will use p-program and program interchangeably, and similarly for p-rule and rule. A p-rule with the empty body is called a *fact*. For a fact, the associated triplet would be of the form $\langle f_d, \_, \_ \rangle$, where "_" indicates the choice of conjunction (or propagation) functions is immaterial. In fact, the conjunction function is not even needed for facts, and any "reasonable" propagation function f_p would return the same result. (See the postulates below.)

The combination functions used in rule based systems are assumed, implicitly or explicitly, to possess certain "reasonable" properties, defined as follows. For simplicity, in our formulation of these properties, we model such a function as a binary mapping on $\mathcal{T}$, when appropriate. This is quite meaningful, upon noting that these functions are assumed to be associative and commutative, as follows.

1. *Monotonicity*: $f(\alpha_1, \alpha_2) \preceq f(\beta_1, \beta_2)$, whenever $\alpha_i \preceq \beta_i$, for $i \in \{1, 2\}$.
2. *Continuity*: f is continuous (in the sense of Scott topology) w.r.t. each one of its arguments.

[2] As a consequence of this function definition and for "consistency" reason, we assume that the disjunction function associated with a predicate symbol p defined by every rule in a p-program is unique.

oarieli@mta.ac.il

3. *Bounded-Above*: $\forall \alpha_1, \alpha_2 \in \mathcal{T}$: $f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i = 1, 2$.
4. *Bounded-Below*: $\forall \alpha_1, \alpha_2 \in \mathcal{T}$: $f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i = 1, 2$.
5. *Commutativity*: $f(\alpha_1, \alpha_2) = f(\alpha_2, \alpha_1)$, $\forall \alpha_1, \alpha_2 \in \mathcal{T}$.
6. *Associativity*: $f(\alpha_1, f(\alpha_2, \alpha_3)) = f(f(\alpha_1, \alpha_2), \alpha_3)$, $\forall \alpha_1, \alpha_2, \alpha_3 \in \mathcal{T}$.
7. $f(\{\!|\alpha|\!\}) = \alpha$, $\forall \alpha \in \mathcal{T}$.
8. $f(\dot{\emptyset}) = \bot$, where $\bot$ is the least element in $\mathcal{T}$.
9. $f(\dot{\emptyset}) = \top$, where $\top$ is the greatest element in $\mathcal{T}$.
10. $f(\alpha, \top) = \alpha$, $\forall \alpha \in \mathcal{T}$.

We require that (i) *conjunction functions satisfy properties 1, 2, 3, 5, 6, 7, 9, 10;* (ii) *propagation functions satisfy properties 1, 2, 3, 10; and* (iii) *disjunction functions satisfy properties 1, 2, 4, 5, 6, 7, 8.*

A brief explanation for these postulations is as follows. This might help identify opportunities to adapt and apply the proposed method in other fixpoint computations with aggregation, in general. The continuity of the combination functions is required in proving the continuity of the fixpoint operator T_P (defined below). The commutativity and associativity of conjunction functions were required for allowing a query optimizer to perform, e.g., subgoal reordering, if desired. The commutativity and associativity of the disjunction functions were needed so that the certainty obtained for each ground atom at each iteration is unique and is independent of the order in which the facts are derived and/or the certainties are combined in that iteration. Boundedness assumptions are imposed in order that derivations make intuitive sense. Property 9 of a conjunction function together with property 10 of a propagation function allows the derivation of a ground atom A with certainty α from the fact $A \xleftarrow{\alpha}$. Property 8 has a similar rationale for disjunction functions. For practical reasons, we further assume that every combination function can be computed efficiently. In theory, we also assume that every such function can be computed with arbitrary precision.

In order to handle disjunction functions which might be sensitive to duplicates, the semantics of the parametric framework is based on multisets. We use $\{\!|\ldots|\!\}$ to represent multisets, and use $\dot{\emptyset}$ to denote the empty multiset. The declarative semantics of p-programs is defined as follows [10]. Let P be a p-program, and B_P be the Herbrand base of P. A *valuation* v of P is a mapping from B_P to $\mathcal{T}$, which assigns to every ground atom in B_P, a certainty value in $\mathcal{T}$. We denote the set of all valuations of P as Υ_P. A *ground instance* of a p-rule r in P is a rule obtained from r by replacing all occurrences of each variable in r with an element of the Herbrand domain. As in Datalog [1], since we do not allow function symbols in p-programs, the Herbrand domain of P will be finite as it would contain only constant symbols. The *Herbrand instantiation* of P, denoted as P^*, is the collection of all ground instances of all p-rules in P.

Let P^* denote the ground instantiation of P and $\rho \equiv (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f_d, f_p, f_c \rangle) \in P^*$ be any instance of rule r in P. The notion of *satisfaction* is defined as follows. We say that:
(1) v satisfies ρ, denoted as $\models_v \rho$, iff $f_p(\alpha_r, f_c(\{\!|v(B_1), \ldots, v(B_n)|\!\})) \preceq v(A)$.
(2) v satisfies r, denoted as $\models_v r$, iff $\models_v \rho$, for every ground instance ρ of r.
(3) v satisfies P, denoted as $\models_v P$, iff $\forall r \in P, \models_v r$, and $\forall A \in B_P$, $f_d(X) \preceq v(A)$,

oarieli@mta.ac.il

where $X = \{\!\{f_p(\alpha_r, f_c(\{\!\{v(B_1), \ldots, v(B_n)\}\!\}))\}\!\} \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f_d, f_p, f_c\rangle) \in P^*\}\!\}$, and $f_d = Disj(\pi(A))$.

The ordering $\preceq$ on $\mathcal{T}$ is extended to valuations in the well-known manner: for any valuations u and v of P, $v \preceq u$ iff $v(A) \preceq u(A)$, for all $A \in B_P$. For all valuations u, v of P and for all $A \in B_P$, we have (1) $(u \otimes v)(A) = u(A) \otimes v(A)$ and (2) $(u \oplus v)(A) = u(A) \oplus v(A)$. We have shown that $\langle \Upsilon_P, \otimes, \oplus\rangle$ is a complete lattice, in which $v_\perp$ is the least element mapping every atom in B_P to $\perp \in \mathcal{T}$, and $v_\top$ is the greatest element mapping every atom to $\top \in \mathcal{T}$. The declarative semantics of a p-program is defined as the least valuation satisfying P, i.e., $\otimes\{v \mid v \in \Upsilon_P \text{ and } \models_v P\}$.

The fixpoint theory developed for p-programs is based on the notion of the *immediate consequence operator* T_P, defined as a mapping from Υ_P to Υ_P, such that for every $v \in \Upsilon_P$ and every atom $A \in B_P$: $T_P(v)(A) = f_d(X)$, where $f_d = Disj(\pi(A))$, and X is the multiset of certainties of A defined as:
$X = \{\!\{f_p(\alpha_r, f_c(\{\!\{v(B_1), \ldots, v(B_n)\}\!\}))\}\!\} \mid (A \xleftarrow{\alpha_r} B_1, \ldots, B_n; \langle f_d, f_p, f_c\rangle) \in P^*\}\!\}$.
The bottom-up iterations of T_P is defined similar to the standard case.

We established that T_P is both monotone and continuous, and that for any p-program P, the least fixpoint of T_P, denoted $lfp(T_P)$, is equivalent to the declarative semantics of P. That is, $lfp(T_P) = \otimes\{v \mid \models_v P\}$

4 Classification of Disjunction Functions

In this section, we study disjunction functions which is essential for understanding the difficulties of evaluating some logic programs with uncertainty. In [10], we classified the family $\mathcal{F}_d$ of disjunction functions into three types, called *types 1–3*, defined as follows.

Definition 4.1. *Let f_d be a disjunction function in the parametric framework. Then, we say*

(1) f_d is of type 1 *provided, $f_d = \oplus$, i.e., when f_d coincides with the lattice join.*
(2) f_d is of type 2 *provided, $\oplus(\alpha, \beta) \prec f_d(\alpha, \beta) \prec \top$, for all $\alpha, \beta \in \mathcal{T} - \{\perp, \top\}$.*
(3) f_d is of type 3 *provided, $\oplus(\alpha, \beta) \prec f_d(\alpha, \beta) \preceq \top$, for all $\alpha, \beta \in \mathcal{T} - \{\perp, \top\}$.*

Intuitively, a type 1 disjunction function means that it coincides with the join $\oplus$ operator in the underlying certainty lattice $\mathcal{T}$, while type 2 functions are strictly greater than $\oplus$, whenever all its arguments are different from the bottom and top elements of the certainty lattice $\mathcal{T}$. A type 3 disjunction function at some input arguments behaves like join and is strictly greater than join at other points. All these types of disjunction functions are assumed to satisfy the postulates introduced above. We emphasize the crucial role of disjunction functions in the problem we address in this paper and the solution technique proposed. On the other hand, conjunction and propagation functions introduce no concern in this regard.

In the context of the parametric framework, we used the above classification to study query optimization [10] and complexity of query evaluations [22]. In both studies, our focus was on disjunction functions of types 1 and 2, whose "behaviours" correspond to many known practical disjunction functions.

oarieli@mta.ac.il

5 A Semi-naive Fixpoint Algorithm

In this section, we propose a bottom-up semi-naive method for evaluating query programs with uncertainty, and establish its equivalence with the corresponding naive evaluation method. First, we provide a quick review of the naive method for p-programs. Even though our motivation for this work is to support disjunction functions of types 2, the proposed solution is applicable to p-programs with disjunction functions of any types (provided they satisfy the postulates).

In a bottom-up naive method, every atom is initially assigned the least certainty value, $\perp$, i.e., every atom is initially assumed to be false. At each iteration, we fire all facts and all applicable rules, and collect the atom-certainty pairs derived as a multiset, noting that when applying a rule, we use the best certainty of each subgoal in the rule body to compute the certainty of the rule head. More precisely, the certainties derived at iteration i for atom A are collected as a multiset $M_i(A)$. At the end of iteration i, $M_i(A)$ is combined into a single certainty for A. This is done by using the disjunction function f_d associated with the predicate symbol of A, as specified in the p-program. This evaluation goes on to some iteration, possibly ω, at which no atom is derived with a "better" certainty. The evaluation method just described corresponds to the fixpoint computation developed by known logic frameworks proposed, obtained by a "straightforward" extension of the standard case, by taking into account the presence of certainties. The naive method is shown in Fig. 1.

Procedure: Naive (P, D; $lfp(T_{P\cup D})$)
  forall $A \in B_P$
1 $v_0(A) := \perp$;
2 $M_1(A) := \{\!\{\alpha \mid (A:\alpha) \in D\}\!\}$;
3 $v_1(A) := f_d(M_1(A))$, where $f_d = Disj(\pi(A))$.
  end forall;
4 $New_1 := \{A \mid (A:\alpha) \in D\}$; $i := 1$;
  while $(New_i \neq \emptyset)$
5 $i := i + 1$;
  forall $(r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f_d, f_p, f_c\rangle) \in P^*$:
6 $M_i(A) := \{\!\{f_p(\alpha_r, f_c(\{\!\{v_{i-1}(B_1), \ldots, v_{i-1}(B_n)\}\!\}))\}\!\}$;
7 $v_i(A) := f_d(M_i)$, where $f_d = Disj(\pi(A))$;
  end forall;
8 $New_i := \{A \mid A \in B_P, v_i(A) \succ v_{i-1}(A)\}$;
  endwhile;
9 $lfp(T_{P\cup D}) := v_i$;
end procedure

Fig. 1. A naive algorithm for evaluating programs with uncertainty.

We next extend the basic naive evaluation method described above and develop a semi-naive algorithm which produces the same result. The basic idea employed here is that we associate with every ground atom A, a pair $\langle M_i, \sigma_i\rangle$, where M_i is a multiset which includes all certainties of A derived so far since the

oarieli@mta.ac.il

initial step, and σ_i is A's certainty obtained by "combining" the certainties in M_i. That is, $\sigma_i = f_d(M_i)$, where $f_d = Disj(\pi(A))$. If at iteration $i+1$, there is an applicable instance of rule r which A as a subgoal, then σ_i is used as the best certainty of A in deriving the certainty of the rule head. Each element of the multiset $M_i(A)$ is of the form $r : \alpha$, indicating a derivation of A with certainty α from rule r. It is important also to note that the use of multiset is crucial, as there could be multiple derivations of the same atom from the same rule in an iteration. For instance, there will be two derivations of $p(1,3)$ with the same certainty from the rule $p(X,Y) \xleftarrow{\alpha} e(X,Z),\ e(Z,Y)$, and the facts $e(1,2)$, $e(2,3)$, $e(1,4)$, and $e(4,3)$, all with the same certainty.

The algorithm is presented in Fig. 2, which takes a collection D of atom-certainty pairs, (basically, the extensional database (EDB), and a collection P of p-rules, and produces the least model of the program, when the fixpoint of $T_{P\cup D}$ is reached, which is captured as the valuation v_i. We denote the multiset operation of union as $\dot{\cup}$ and the multiset difference as $\dot{-}$, both of which respect duplicates.

Procedure: Semi-Naive (P, D; $lfp(T_{P\cup D})$)
  forall $A \in B_P$:
1 $v_0(A) := \bot$;
2 $M_1(A) := \{\!\{\alpha \mid (A:\alpha) \in D\}\!\}$;
3 $v_1(A) := f_d(M_1(A))$, where $f_d = Disj(\pi(A))$;
  end forall;
4 $New_1 := \{A \mid (A:\alpha) \in D\}$; $i := 1$;
  while $(New_i \neq \emptyset)$
5 $i := i+1$;
6 **forall** $A \in B_P$:
    if $\exists (r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f_d, f_p, f_c\rangle) \in P^*$
    such that $\exists\, B_j \in New_i$, for some $j \in \{1,\ldots,n\}$:
    then begin
7 $M_i(A) := M_{i-1}(A)$;
8 **forall** $(r : A \xleftarrow{\alpha_r} B_1, \ldots, B_n;\ \langle f_d, f_p, f_c\rangle) \in P^*$
      such that $B_j \in New_i$, for some $j \in \{1,\ldots,n\}$:
9 $M_i(A) := M_i(A) \dot{-} \{\!\{\sigma^r_{i-1}(A)\}\!\} \dot{\cup} \{\!\{\sigma^r_i(A)\}\!\}$, where
        $\sigma^r_i(A) := f_p(\alpha_r, f_c(\{\!\{v_{i-1}(B_1), \ldots, v_{i-1}(B_n)\}\!\}))$;
      end forall;
10 $v_i(A) := f_d(M_i(A))$, where $f_d = Disj(\pi(A))$;
    end;
11 **else** $v_i(A) := v_{i-1}(A)$;
  end forall;
12 $New_i := \{A \mid A \in B_P, v_i(A) \succ v_{i-1}(A)\}$;
13 **endwhile**
14 $lfp(T_{P\cup D}) := v_i$;
end procedure

Fig. 2. A semi-naive algorithm for evaluating programs with uncertainty.

oarieli@mta.ac.il

Initially in line 1, every atom is associated with $\langle \dot{\emptyset}, \perp \rangle$. In lines 2 and 3, we then revise the certainties of the atoms given as EDB facts. We also need to keep track of those atom-certainty pairs, where the certainty of the atom was increased compared to the previous iteration. This is done through the set New_i defined initially in line 4 and revised subsequently in line 12. At iteration $i+1$, we select and apply those rules r, such that r has at least one subgoal B appearing in New_i, i.e., the overall certainty of B was raised at iteration i. This is done in line 6 in the algorithm. When r is applied, the certainty of subgoal B used in this rule application is $\sigma(M_i(B))$, shown in lines 3 and 10. At step $i+1$, for every ground atom $A : \gamma$ derived by r, we remove from $M_{i+1}(A)$, every derivation of A by r, and add certainty $r : \gamma$ to this multiset [*L*9]. In line 10, we compute the new overall certainty of A using the disjunction $Disj(\pi(A))$ associated with the predicate symbol of atom A. For atoms whose certainty did not increase, the valuation $v_i(A)$ defined at iteration i would be the same as the old one, $v_{i-1}(A)$. This is done in line 11. After all applications of possible rules at iteration i, we are able to define the set of New_i of atoms whose certainties are increased. This is done in line 12. If this set is empty, then the evaluation terminates and the result v_i is returned, as shown in line 14. Otherwise, the execution of the algorithm at the while loop continues.

A crucial statement to note is the multiset of certainties collected in line 9, which is a basis for correctness of this algorithm, making the result at every iteration to be identical to the naive method. It ensures that no derivation of A from the same rule r at iteration i is mixed with the derivations of A by r at earlier iterations. This is the problem we observed with the standard semi-naive technique resulted in wrong computation, as explained in section 2. This maybe stated as a *correctness requirement* as follows. "If disjunction functions of types 2 and 3 are present in a p-program, an evaluation procedure should combine the certainties of atoms derived at the same iteration, and not combine newly derived certainties with prior certainties of the same atom from the same rule."

A point of avoiding redundancy is that we do not need to keep track of all previous derivations of A by r; it is enough to consider every derivations of A obtained at the previous iteration. The basis for this is as follows. If r is not a recursive rule, then in a semi-naive evaluation, it will be applied only once to derive an atom from a rule instance. On the other hand, if r is recursive rule which derived atom A at iteration i, this rule will continue derive A at every subsequent iteration. In this case, if i is the first time we derived A by r, the multiset difference operation in line 9 does nothing. Otherwise, this line ensures the above correctness requirement. This is a key observation which also allows efficient implementation of this method in terms of time and memory requirements. The above arguments provide a basis for the proof of the following result.

Theorem 5.1. *Let P be any p-program in the parametric framework and D be a collection of facts. A fixpoint computation of P over D using the semi-naive method proposed above produces the same result as the naive method.*

oarieli@mta.ac.il

Let us consider the program P_1 of section 2 again to explain the steps of this algorithm. Initially, every ground atom is assigned the certainty 0. At iteration 1 we obtain $I_1 = \{r_1 : B : 0.5,\ r_2 : C : 0.8\}$ from r_1 and r_2. At iteration 2, we apply r_3, and hence $I_2 = \{r_3 : A : 0.8,\ r_1 : B : 0.5,\ r_2 : C : 0.8\}$. At iteration 3, we apply r_4 and use 0.8 as the certainty of A in the rule body. This yields A with certainty $0.6{\times}0.5{\times}0.8 = 0.24$. The multiset $M_3(A)$ associated with A at iteration 3 would thus include $r_3 : 0.8$ and $r_4 : 0.24$, and the certainty of A at this iteration is improved to $ind(0.8, 0.24) = 0.848$. At iteration 4, we apply r_4 again, using 0.848 in the rule body as the certainty of A. This yields $r_4 : A : 0.2544$, which replaces the derivation $r_4 : A : 0.24$ in $M_4(A)$. Therefore, $M_4(A) = \{\!\{0.8, 0.2544\}\!\}$, and hence A's certainty at iteration 4 would be $ind(0.8, 0.2544) = 0.85088$. Since the certainty of A improves at every iteration, this computation will terminate only in the limit. Since the certainty values obtained and combined at every iteration for each atom are identical with those obtained in the naive method, the semi-naive method produces the same result as the naive method.

It is not hard to convince ourselves that the proposed method extends the standard fixpoint evaluation upon noting that the disjunction function in the standard framework is *max*, indicating that duplicate values collected as a multiset in line 9 of the algorithm may be ignored by collecting them as a set.

6 Implementation and Experimental Results

In order to determine the practical merits of the proposed algorithm, we developed a system prototype in C++ which can evaluate a program with uncertainty using either the naive method or the semi-naive, determined by the user. To measure the efficiency of the proposed technique, we conducted a number of experiments of evaluating p-programs and compared the execution time of these two techniques. For this, we considered two classes of logic program with uncertainty computing the transitive closure of a binary predicate p. The first p-program P_2 includes r_1 and r_2 defined below and the second program P_3 includes r_1 and r_3. Note that P_2 is a linear (recursive) program whereas P_3 is a double recursive program.

$$
\begin{array}{ll}
r_1 : p(X,Y) \xleftarrow{1} e(X,Y); & \langle ind, \times, \_ \rangle. \\
r_2 : p(X,Y) \xleftarrow{1} e(X,Z),\ p(Z,Y); & \langle ind, \times, \times \rangle. \\
r_3 : p(X,Y) \xleftarrow{1} p(X,Z),\ p(Z,Y); & \langle ind, \times, \times \rangle.
\end{array}
$$

We considered a collection of EDB facts with different number of tuples. Each collection in our experiment included cyclic data of the form: $e(0,1),\ e(1,2),\ \cdots,\ e(n,0)$, for $n = 10K$, where $1 \leq K \leq 15$. As with the rules, the certainty associated with every tuple was set to 1. Our choice of this certainty value because we wanted to compare the "timing" with timing of an existing powerful deductive database such as CORAL [19], even though the actual certainties computed would not be identical, in general, if the certainties of the rule were different than the value 1 we chose.

oarieli@mta.ac.il

Table 1. Summary of our experiment results for program P_2.

Number of EDB Tuples	Number of IDB Tuples	Number of Iterations	Time (Sec) Naive	Time (Sec) Semi-Naive	Speed-up (N to SN)	Time (Sec) CORAL
10	100	11	0	0.07	0	1.21
20	400	21	2	0.39	5.1	4.84
30	900	31	11	1.4	7.9	10.79
40	1,600	41	36	3.69	9.8	19.35
50	2,500	51	92	8.15	11.3	30.31
60	3,600	61	198	15.97	12.4	42.29
70	4,900	71	392	27.61	14.2	60.19
80	6,400	81	715	45.17	15.2	78.28
90	8,100	91	1246	69.83	17.8	96.79
100	10,000	101	2,010	103.10	19.5	120.99
110	12,100	111	3,141	142.96	22	146.70
120	14,400	121	4,787	193.59	24.5	171.12
130	16,900	131	7,090	257.53	27.5	209.10
140	19,600	141	10,059	341.50	29.5	241.52
150	22,500	151	14,114	446.68	31.6	279.57

To conduct the experiments, we used a desktop computer with 2 CPUs (Ultra Sparc-II) running at 296 MHz, with 100 MB of RAM and 4.2 GB hard disk, under the Solaris 8 operating system. The experimental results are illustrated in Tables 1 and 2. Table 1 shows the time measured in seconds for evaluating query $p(X, Y)$ on the program P_2 with different EDB sizes, identified in the first column, in terms of the number of tuples. The 4th column in these tables includes the timing of the basic naive method with uncertainty and column 5 includes the timing of the semi-naive. Column 6 in Table 1 shows the speed-up achieved by the semi-naive method over the naive method, obtained by dividing the corresponding time measures of the naive to the semi-naive case. We can see from this table that the speed-up achieved on average is about 17 times. This is a significant improvement upon noting that the index structure developed was not sophisticated. Finally, the last column in Table 1 shows the time it took to evaluate the standard transitive closure program in CORAL with the corresponding EDB facts. As we can see, CORAL outperforms our prototype system with a factor of 1.5 times on large EDB sets of 120 tuples or more, which is not a bad result as our first implementation attempt. Table 2 includes the timing we obtained in evaluating the double recursive program P_3 in an attempt to assess the efficiency of our system prototype. The average speed-up we got in this case is about 14. As we can see, CORAL is 6 times faster for the data set of size 120 tuples. Recall that in this case, CORAL evaluates the double recursive program in standard logic without having to deal with uncertainty values and functions.

oarieli@mta.ac.il

Table 2. Summary of our experiment results for program P_3.

Number of EDB Tuples	Number of IDB Tuples	Time (Sec) Naive	Time (Sec) Semi-Naive	Speed-up (N to SN)	Time (Sec) CORAL
10	100	1	0.16	6.3	0.07
20	400	7	1.11	6.3	0.49
30	900	28	3.33	8.4	1.46
40	1600	132	12.71	10.4	3.39
50	2500	285	24.59	11.6	6.56
60	3600	566	44.2	12.8	11.26
70	4900	1765	121.48	14.5	18.39
80	6400	2809	177.79	15.8	26.77
90	8100	4297	246.28	17.4	37.91
100	10000	6420	335.02	19.2	51.99
110	12100	9659	436.89	22.1	69.02
120	14400	14482	568.97	22.5	91.92

7 Discussions

In this section, we discuss when the proposed method could/should be used as an alternative to existing evaluation methods. Our objective here is to provide some intuitive arguments in a rather informal way.

An important question at this point is that: *when can we use existing bottom-up (or top-down) inference systems to evaluate IB programs with uncertainty?* In fact, our first attempt to implement the parametric framework was to use Coral [19] and XSB [20], both of which are powerful and efficient systems, and support multisets, among other interesting and features. We could successfully implement the IB frameworks which use sets as the basis of their semantics structure. A version of Coral was provided to us which also supports an annotation, called `@aggsel_per_iteration`, which would be useful in our context if we could change the default evaluation method of Coral to be naive, but then it would be inefficient when the EDB is large.

Returning to our question that when we may take advantage of existing efficient engines, we note that there are three factors influential here. (1) the program structure, being recursive or otherwise, (2) the data being "cyclic" or not, and (3) existence of a recursive predicate in the program which is associated with a strict disjunction f_d, as defined in section 4. The third condition holds when $f_d(\alpha, \beta) \succ \oplus\{\alpha, \beta\}$, whenever α and β are both different than $\bot$ and $\top$. This intuitively means that the result of f_d is always "better" than its arguments. If any of these conditions fails to hold, we can luckily use the existing systems to evaluate such programs. On the other hand, our algorithm may always be used, in particular when all the above conditions hold. In this case, the book keeping in our algorithm is required only for recursive predicates which are associated with a strict disjunction such as f_d.

oarieli@mta.ac.il

We also remark that using a combination of the rule id and atom pair of the form (r_i, A) as an alternative technique to implement the desired computation while avoiding the complication of dealing with multisets will not work since in this case, we cannot distinguish two (or more) derivations of the same atom by the same rule in the same iteration step. For instance, consider a program with the rules: $r_1 : p(A, B) \stackrel{\alpha}{\longleftarrow} e(A, B)$ and $r_2 : p(A, B) \stackrel{\beta}{\longleftarrow} e(A, C),\ p(C, B)$, and the facts $e(1, 2),\ e(1, 3),\ e(2, 4)$, and $e(3, 4)$, all with certainty 1. In this example, using the pair $r_2, p(1, 4)$ to annotate derivations of $p(1, 4)$ by r_2 will result in loosing one of the two and hence wrong computation, if the disjunction function associated with predicate p is e.g., *ind*.

We believe that the proposed method can be adapted in some more general context of fixpoint computation with aggregation in which it is important to collect derivations as multisets and in which the aggregation functions used are strictness and subject to the same properties as disjunction functions in the parametric framework. Also, even though our focus in this work was on the IB frameworks, the method developed can be adapted to AB frameworks as well.

8 Concluding Remarks

In this paper, we studied evaluations of logic programs and deductive databases in the context of the parametric framework over the complete lattice $[0, 1]$. We assumed that arithmetic computations over reals could be carried out with arbitrary precision. We illustrated that the standard semi-naive evaluation method does not have a counterpart in logic programs with uncertainty. Our motivation in this work was to fill this gap and proposed a desired evaluation method which is equivalent to the naive method. Our implementation of the proposed method and the experimental results obtained show that the proposed method together with some basic heuristics we used in the construction of a hash-based structure in the prototype system developed support a desired, efficient fixpoint computation with uncertainty. We are currently pursuing some ideas to further improve the efficiency of the system. Preliminary results in this direction are encouraging. In other directions, we would like to extend our method to handle negation in the parametric framework [25,26].

Acknowledgements. The authors are grateful to anonymous referees for their helpful comments.

References

1. Ceri S., Gottlob G., and Tanca L. *Logic programming and Databases.* Berlin, New York: Springer-Verlag, 1990.
2. Dubois Didier, Lang Jérôme, and Prade Henri. Towards possibilistic logic programming. In *Proc. 8th Intl. Conf. on Logic Programming*, pages 581–596, 1991.

oarieli@mta.ac.il

3. Fitting M.C. Logic programming on a topological bilattice. *Fundamenta Informaticae*, 11:209–218, 1988.
4. Fitting M.C. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
5. Kifer, M. and Li, A. On the semantics of rule-based expert systems with uncertainty. In M. Gyssens, J. Paradaens, and D. van Gucht, editors, *2nd Intl. Conf. on Database Theory*, pages 102–117, Bruges, Belgium, August 31-September 2 1988. Springer-Verlag LNCS-326.
6. Kifer M. and Subrahmanian V.S. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
7. Lakshmanan, Laks V.S. An epistemic foundation for logic programming with uncertainty. In *Proc. 14th Conf. on the Foundations of Software Technology and Theoretical Computer Science (FST and TCS'94)*. Springer-Verlag, LNCS-880, December 1994.
8. Lakshmanan, Laks V.S. and Sadri, F. Probabilistic deductive databases. In *Proc. Intl. Logic Programming Symposium*, pages 254–268, Ithaca, NY, November 1994. MIT Press.
9. Lakshmanan Laks V.S. and Sadri F. Modeling uncertainty in deductive databases. In *Proc. Intl. Conf. on Database Expert Systems and Applications (DEXA '94)*, Athens, Greece, September 1994. Springer-Verlag, LNCS-856.
10. Lakshmanan, Laks V.S. and Shiri, Nematollaah. A parametric approach to deductive databases with uncertainty. In *Proc. Intl. Workshop on Logic in Databases (LID'96)*, pages 61–81, San Miniato, Italy, July 1996. Springer-Verlag, LNCS-1154.
11. Lakshmanan, Laks V.S. and Shiri, Nematollaah. Logic programming and deductive databases with uncertainty: A survey. In *Enclyclopedia of Computer Science and Technology*, volume 45, pages 153–176. Marcel Dekker, Inc., New York, 2001.
12. Lakshmanan Laks V.S. and Shiri Nematollaah. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
13. Leach Sonia M. and Lu James J. Query processing in annotated logic programming: Theory and implementation. *Journal of Intelligent Information Systems*, 6(1):33–58, January 1996.
14. Lloyd J. W. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.
15. Ng R.T. and Subrahmanian V.S. Relating Dempster-Shafer theory to stable semantics. Tech. Report UMIACS-TR-91-49, CS-TR-2647, Institute for Advanced Computer Studies and Department of Computer Science University of Maryland, College Park, MD 20742, April 1991.
16. Ng R.T. and Subrahmanian V.S. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, December 1992.
17. Ng R.T. and Subrahmanian V.S. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Automated Reasoning*, 10(2):191–235, 1993.
18. Parsons, S. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353–372, 1996.
19. Ramakrishnan R., Srivastava D., and Sudarshan S. CORAL: Control, relations, and logic. In *Proc. Intl. Conf. on Very Large Databases*, 1992.
20. Sagonas Konstantinos, Swift Terrance, and Warren David S. XSB as an efficient deductive database engine. In *Proc. of the ACM SIGMOD Intl. Conf. on the Management of Data*, pages 442–453, Minneapolis, Minnesota, May 1994.

oarieli@mta.ac.il

21. Shapiro E. Logic programs with uncertainties: a tool for implementing expert systems. In *Proc. IJCAI'83*, pages 529–532. William Kaufmann, 1983.
22. Shiri, Nematollaah. *Towards a Generalized Theory of Deductive Databases with Uncertainty*. PhD thesis, Department of Computer Science, Concordia University, Montreal, Canada, August 1997.
23. Subrahmanian V.S. On the semantics of quantitative logic programs. In *Proc. 4th IEEE Symposium on Logic Programming*, pages 173–182, Computer Society Press, Washington DC, 1987.
24. van Emden M.H. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4(1):37–53, 1986.
25. Yann Loyer and Umberto Straccia. The well-founded semantics in normal logic programs with uncertainty. In *Proc. of the 6th Int'l Symp. on Functional and Logic Programming (FLOPS '02), LNCS 2441*, pages 67–78. Springer Verlag, 2002.
26. Yann Loyer and Umberto Straccia. Default knowledge in logic programs with uncertainty. In *Proc. of the 19th Int'l Conf. on Logic Programming (ICLP '03), Mumbai, India*, Dec. 9–13, 2003.

oarieli@mta.ac.il

Towards a Generalized Interaction Scheme for Information Access

Yannis Tzitzikas[1], Carlo Meghini[2], and Nicolas Spyratos[3]

[1] Information Technology, VTT Technical Research Centre of Finland
ext-yannis.tzitzikas@vtt.fi
[2] Istituto di Scienza e Tecnologie dell' Informazione [ISTI], CNR, Pisa, Italy
meghini@isti.cnr.it
[3] Laboratoire de Recherche en Informatique, Universite de Paris-Sud, France
spyratos@lri.fr

Abstract. We introduce the formal framework of a generalized interaction scheme for information access between users and information sources. Within this framework we describe an *interaction manager* which supports more complex interaction schemes than those that are supported by existing systems, including: query by example, answer enlargement/reduction, query relaxation/restriction, index relaxation/contraction, "relevance" feedback, and adaptation facilities. We give the foundations of this interaction manager from a mathematical point of view, in terms of an abstract view of an information source.

1 Introduction

Information sources such as information retrieval systems [2], or databases and knowledge bases [12], aim at organizing and storing information in a way that allows users to retrieve it in a flexible and efficient manner. Commonly, for retrieving the desired information from an information source, the user has to use the query language that is provided by the system.

We propose an interaction scheme whose objective is to make the desired objects easy to find for the user, even if the source has a query language which is *unknown* to the user. This scheme is actually a generalization of the interaction schemes that are currently used by information systems. In particular, we describe an *interaction manager* which supports several kinds of interaction, including: query by example, index relaxation/contraction, query relaxation/restriction, answer enlargement/reduction, "relevance" feedback, and adaptation facilities, in a uniform manner.

We view the interaction of a user with the information source as a sequence of *transitions* between *contexts* where a context is a consistent "interaction state". The user has at his/her disposal several ways to express the desired transition. Then, it is the interaction manager that has to find (and drive the user to) the new context. Methods allowing the user to specify a transition relatively

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 291–301, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

to the current context are also provided. Furthermore, we describe methods for restricting the set of transitions to those that can indeed lead to a context. As we shall see below, the unified interaction scheme that we introduce allows defining more complex interaction mechanisms than those that are supported by existing systems. We describe this scheme in terms of an abstract view of a source.

The paper is organized as follows. Section 2 introduces contexts and context transitions. Section 3 describes how context transitions can be specified using replacements. Section 4 describes how the interaction manager can find a new context after a context transition specification. Section 5 introduces relative replacements and Section 6 describes context transitions using restricted relative replacements. Section 7 concludes the paper and identifies issues for further research.

2 A Context-Based Interaction Scheme

We view a source S as a function $S : Q \rightarrow \mathcal{A}$ where Q is the set of all queries that S can answer, and $\mathcal{A}$ is the set of all answers to those queries, i.e. $\mathcal{A}$=$\{\ S(q) \mid q \in Q\}$. As we focus on retrieval queries, we assume that $\mathcal{A}$ is a subset of $\mathcal{P}(Obj)$, the powerset of Obj, where Obj is the set of all objects stored at the source.

Let $\mathcal{S}$ be the set of all sources that can be derived by "updating" the source S (e.g. for adapting it), but for the moment let us suppose that $\mathcal{S}$ is the set of all functions from Q to $\mathcal{P}(Obj)$.

Let $\mathcal{U}$ denote the set of all triples in $\mathcal{S}\times Q \times \mathcal{A}$, i.e. $\mathcal{U} = \mathcal{S}\times Q \times \mathcal{A}$. A triple $c = (S, q, A) \in \mathcal{U}$ is called an interaction context, or *context* for short, if $S(q) = A$. Let $\mathcal{C}$ denote the set of all contexts, i.e. $\mathcal{C}$= $\{\ (S, q, A) \in \mathcal{U}\ | S(q) = A\}$. Given a context $c = (S, q, A)$, S is called the *source view* of c, q is called the *query* of c and A is called the *answer* of c. The interaction between the user and the source is carried out by a software module called *Interaction Manager* (IM). We view the interaction of a user with the source as a sequence of *transitions* between contexts. At any given time, the user is in one context, the *focal context*. At the beginning of the interaction with the system the user starts from the initial context $(S, \epsilon, \emptyset)$ where S is the stored information source, ϵ is the empty query and $\emptyset$ is the empty answer[1]. There are several methods the user can use for moving from one context to another, i.e. for changing the focal context. For example, in the traditional query-and-answer interaction scheme, the user actually "replaces" the current query q by formulating a new query q' and the "interaction manager" drives him to the new context (S, q', A') where $A' = S(q')$. However this is only one way of changing the focal context. Several other ways will be presented below.

At first we introduce some preliminary material. There are three partially ordered sets (posets) that can be exploited by the interaction manager:

[1] We assume that $S(\epsilon) = \emptyset$ therefore $(S, \epsilon, \emptyset)$ is a context.

oarieli@mta.ac.il

– The poset of answers $(\mathcal{P}(Obj), \subseteq)$

– The poset of queries $(Q, \leq)$. Given two queries q and q' of Q, we write $q \leq q'$ iff $S(q) \subseteq S(q')$ in every possible source S in $\mathcal{S}$. We write $q \sim q'$ is both $q \leq q'$ and $q' \leq q$ hold. Let $Q_\sim$ denote the set of equivalence classes induced by $\sim$ over Q.

– The poset of sources $(\mathcal{S}, \sqsubseteq)$. Given two sources S and S' in $\mathcal{S}$, $S \sqsubseteq S'$ iff $S(q) \subseteq S'(q)$ in every query $q \in Q$.

For every element x of the above lattices, we shall use $Br(x)$ to denote the elements that are greater than or equal to x, and $Nr(x)$ to denote the elements that are less than or equal to x in the corresponding poset.

3 Context Transition Specifications (CTSs) through Replacements

At first we study the case where the user *replaces* one component of the focal context (i.e. either S, q or A) by another component by explicitly providing the new component. Later on, we will further study these replacements and we will describe methods that allow the user to specify the desired replacement without having to provide explicitly the new component (i.e. methods for defining the new component relatively to the current).

The replacements that can be applied to a context $c = (S, q, A)$ can be:

(a) query replacement, denoted by $[q \to q']$,
(b) answer replacement, denoted by $[A \to A']$, and
(c) source view replacement, denoted by $[S \to S']$

As mentioned earlier, the user should always be in a context i.e. in a triple (S, q, A) where $S(q) = A$. This means that after any of the above kinds of replacement, the interaction manager should try to reach a context c' by changing one (or both) of the remaining two components of the focal context. Instead of leaving the IM to decide, it is the user that indicates to the IM the component(s) to be changed during a replacement. A replacement plus an indication of the above kind, is what we call a *Context Transition Specification (CTS)*. Below we discuss each possible CTS, assuming a focal context (S, q, A). For each CTS, we also discuss the motivation beneath. We do it in brief because we mainly focus on founding this interaction scheme from a mathematical point of view. However, we think that the set of CTSs that we propose are elemental, in the sense that other more sophisticated interaction schemes can be analyzed using the set of CTSs that we propose.

Query replacement $[q \to q']$

– $[q \to q'/A]$
Here the answer A must be changed. This is the classical query-and-answer interaction scheme: when the user replaces the current query q with a new

oarieli@mta.ac.il

query q', the user is given a new answer (i.e. an A' such that $A' = S(q')$). Thus we can write: $[q \rightarrow q'/A](S, q, A) = (S, q', S(q'))$.

- $[q \rightarrow q'/S]$
  Here the source view S must be changed. This means that the user replaces the current query q by a query q', because the user wants the current answer A to be the answer of q', not of q. The IM should try to "adapt" to the desire of the user by changing the source view from S to an S' such that $S'(q') = A$.

Answer replacement $[A \rightarrow A']$

- $[A \rightarrow A'/q]$
  Here the query must be changed. This interaction scheme may help the user to get acquainted with the query language of the source. It can be construed as an alternative query formulation process. The user selects a number of objects (i.e. the set A') and asks from the IM to formulate the query that "describes" these objects. Subsequently the user can change the query q' in a way that reflects his/her information need. Roughly this resembles the Query By Example (QBE) process in relational databases. It also resembles the relevance feedback mechanisms in Information Retrieval systems. For example, the user selects a subset A' of the current answer A consisting of those elements of A which the user finds relevant to his/her information need. Subsequently, the IM has to change appropriately the query.
- $[A \rightarrow A'/S]$
  Here the source view S has to be changed. This case resembles the CTS $[q \rightarrow q'/S]$ that was described earlier. Here the user wants A' (instead of A) to be the answer of q. The IM should try to adapt to the desire of the user by changing the source view from S to an S' such that $S'(q) = A'$.

Source view replacement $[S \rightarrow S']$

- $[S \rightarrow S'/A]$
  Here the answer A must be changed. This is the classical interaction scheme: whenever the source changes (e.g. after an update) the answers of queries change as well, i.e. here we have $A' = S'(q)$.
- $[S \rightarrow S'/q]$
  Here the query q must be changed. This resembles the way that a relational database management system changes the query q that defines a relational view, after an update of the database, in order to preserve the contents (tuples) of the view.

We have seen six kinds of context transition specifications. Table 1 summarizes the above discussion and it also includes the cases where the IM can change two components of the focal context during one transition. In summary, the user has 9 different ways to specify the desired context transition.

oarieli@mta.ac.il

Table 1. Replacements and possible reactions of the interaction manager

Replacements on $c = (S, q, A)$		c'		
Query Replacement	$[q \to q']$	$(S, q', \mathbf{A'})$	$(\mathbf{S'}, q', A)$	$(\mathbf{S'}, q', \mathbf{A'})$
Answer Replacement	$[A \to A']$	$(S, \mathbf{q'}, A')$	$(\mathbf{S'}, q, A')$	$(\mathbf{S'}, \mathbf{q'}, A')$
Source View Replacement	$[S \to S']$	$(S', \mathbf{q'}, A)$	$(S', q, \mathbf{A'})$	$(S', \mathbf{q'}, \mathbf{A'})$

4 Finding the Target Context(s)

Given a focal context c and a context transition specification R, the role of the IM is to find the desired context $R(c)$, if one exists.

Searching for an answer (in the cases $[q \to q'/A]$ and $[S \to S'/A]$)

The cases in which the interaction manager has to change the answer in order to reach to a context (i.e. after a query or source replacement), are relatively simple: the desired context always exists and the desired answer A' can be derived using the query evaluation mechanism of the source. In particular, in the case $[q \to q'/A]$ ("query replacement - change of answer") the new context is $(S, q', \mathbf{S}(\mathbf{q'}))$, while in the case $[S \to S'/A]$ ("source replacement - change of answer") the new context is $(S', q, \mathbf{S'}(\mathbf{q}))$.

The cases in which the interaction manager has to find a new query or a new source view are less straightforward. The main problem is that the desired context does not always exist.

Searching for a query q (in the cases $[A \to A'/q]$ and $[S \to S'/q]$)

In cases $[A \to A'/q]$ and $[S \to S'/q]$ we are looking for a $q \in Q$ such that $S(q) = A$ for given S and A. Let us first try to define a "naming service", i.e. a method for computing one or more queries that describe (name) a set of objects $A \subseteq Obj$. For supporting the naming service we would like to have a function $n : \mathcal{P}(Obj) \to Q$ such that for each $A \subseteq Obj$, $S(n(A)) = A$. Having such a function, we would say that the query $n(A)$ is an exact name for the object set A. Note that if S is an *onto* function then the naming function n coincides with the inverse relation of S, i.e. with the relation $S^{-1} : \mathcal{P}(Obj) \to Q$. However, this is not always the case, as more often than not, S is not an onto function, i.e. $\mathcal{A} \subset \mathcal{P}(Obj)$. Furthermore, if S is onto and one-to-one, then S^{-1} is indeed a function, thus there is always a unique q such that $S(q) = A$ for each $A \subseteq Obj$[2].

As S is not always an onto function, we shall introduce two "approximate" naming functions, a *lower* naming function n^- and an *upper* naming function n^+. We can define the function n^- and n^+ as follows:

$$n^-(A) = lub\{\ q \in Q \mid S(q) \subseteq A\}$$
$$n^+(A) = glb\{\ q \in Q \mid S(q) \supseteq A\}$$

[2] Usually, sources are not one-to-one functions. For instance, the supported query language may allow formulating an infinite number of different queries, while the set of all different answers $\mathcal{A}$ is usually finite (e.g. $\mathcal{P}(Obj)$).

oarieli@mta.ac.il

where A is any subset of Obj. Now let A be a subset of Obj for which both $n^-(A)$ and $n^+(A)$ are defined (i.e. the above lub and glb exist). It is clear that in this case it holds:

$$S(n^-(A)) \subseteq A \subseteq S(n^+(A))$$

and that $n^-(A)$ and $n^+(A)$ are the best "approximations" of the exact name of A. Note that if $S(n^-(A)) = S(n^+(A))$ then both $n^-(A)$ and $n^+(A)$ are exact names of R. If Q is a query language that (a) supports disjunction ($\vee$) and conjunction ($\wedge$) and it is closed with respect to both these operations, and (b) has a top ($\top$) and a bottom ($\bot$) element such that $S(\top) = Obj$ and $S(\bot) = \emptyset$, then the functions n^- and n^+ are defined for every subset A of Obj. Specifically, in this case $(Q_\sim, \leq)$ is a complete lattice, thus these functions are defined as:

$$n^-(A) = \bigvee\{\, q \in Q \mid S(q) \subseteq A\}$$
$$n^+(A) = \bigwedge\{\, q \in Q \mid S(q) \supseteq A\}$$

As Q is usually an infinite language, $n^-(A)$ and $n^+(A)$ are queries of infinite length. This means that in practice we also need a method for computing a query of finite length that is equivalent to $n^-(A)$ and another that is equivalent to $n^+(A)$. If however Q does not satisfy the above conditions (a) and (b), then $n^-(A)$ and $n^+(A)$ may not exist. For such cases, we can define n^- and n^+ as follows: $n^-(A) = max\{\, q \in Q \mid S(q) \subseteq A\}$ and $n^+(A) = min\{\, q \in Q \mid S(q) \supseteq A\}$, where max returns the maximal element(s), and min the minimal element(s). Clearly, in this case we may have several lower and upper names for a given A. A specialization of these naming functions for the case of taxonomy-based sources is described in [10].

Let us now return to our problem. If a naming function n_S is available for source S, then the context we are looking for exists and can be found as follows:

- $[A \rightarrow A'/q](S, q, A) = (S, n_S(A'), A')$
- $[S \rightarrow S'/q](S, q, A) = (S', n_{S'}(A), A)$

If only approximate naming functions are available, then two "approximate" next contexts exist:

- $[A \rightarrow A'/q](S, q, A) = \begin{cases} (S, n_S^-(A'), S(n_S^-(A'))) \\ (S, n_S^+(A'), S(n_S^+(A'))) \end{cases}$
- $[S \rightarrow S'/q](S, q, A) = \begin{cases} (S', n_{S'}^-(A), S'(n_{S'}^-(A))) \\ (S', n_{S'}^+(A), S'(n_{S'}^-(A))) \end{cases}$

Notice that in the above cases, IM does not change only q, but also the answer. In the case $[A \rightarrow A'/q]$, the new context has an answer, say A", which is the closest possible to the requested (by the user) answer A'. In the case $[S \rightarrow S'/q]$, the new context has an answer A' which is the closest possible to the current A.

oarieli@mta.ac.il

Searching for a source view S (in the cases $[A \rightarrow A'/S]$ and $[q \rightarrow q'/S]$)
In cases $[A \rightarrow A'/S]$ and $[q \rightarrow q'/S]$ we are looking for a $S \in \mathcal{S}$ such that $S(q) = A$ for given q and A. Note that the desired source S always exists if and only if $\mathcal{S}$ is the set of all functions from Q to $\mathcal{A}$. We cannot describe any mechanism for finding the desired S' within such an abstract framework. However, later on we shall see how restricted relative replacements and a source relaxation/contraction mechanism can be exploited in order to support this kind of interaction.

The following list summarizes how the IM can find the target context after each kind of CTS:

(1) $[q \rightarrow q'/A](c) = (S, q', S(q'))$, i.e. it relies on query evaluation
(2) $[S \rightarrow S'/A](c) = (S', q, S'(q))$, i.e. it relies on query evaluation
(3) $[A \rightarrow A'/q](c) = (S, n_S(A'), A')$, i.e. it relies on naming functions
(4) $[S \rightarrow S'/q](c) = (S', n_{S'}(A), A)$, i.e. it relies on naming functions
(5) $[A \rightarrow A'/S](c) =?$, i.e. this is an open issue
(6) $[q \rightarrow q'/S](c) =?$, i.e. this is an open issue

5 Relative Replacements and Relative CTSs

Until now we have described how the IM could react to a context transition specification. Now we shall focus on the replacements. Certainly, the user can manually specify a replacement, e.g. submit a new query, select a set of objects $A' \subseteq Obj$, or update the source S by adding, deleting or modifying some of the contents of S.

Here we describe methods that allow the user to specify the desired replacement without having to provide explicitly the new component, i.e. methods for defining the new component relatively to the current. These methods can be helpful for the user during the interaction with the system.

The three partial orders mentioned earlier can be exploited for moving to a component (answer, query, or source) that covers, or is covered by, the current component. Given a component x, let x^+ denote a component that covers x, and let x^- denote a component that is covered by x [3]. Note that there may not always exist a unique x^+ or a unique x^-. Specifically, we may have zero, one, or more x^+'s and x^-'s for a given x.

Let $Up(x)$ denote a component among those greater than x and that the IM can compute (ideally $Up(x) = x^+$), and let $Down(x)$ denote a component among those less than x and that the IM can compute (ideally $Down(x) = x^-$). Notice that these Up and $Down$ functions correspond to

- a *query relaxation/contraction* mechanism, if applied to Q,
  (i.e. $[q \rightarrow Up(q)]$ and $[q \rightarrow Down(q)]$)

[3] An element x^+ covers an element x if $x^+ > x$ and $x^+ \leq x'$ for every $x' > x$. An element x is covered by an element x^- if $x^- < x$ and $x^- \geq x'$ for every $x' < x$.

oarieli@mta.ac.il

Such mechanisms have been proposed for several kinds of sources, including relational [6], semi-structured [4], taxonomy-based [11], Description-Logics-based [9,3] and Web sources [7].

- an *answer enlargement/reduction* mechanism, if applied to $\mathcal{A}$, (i.e. $[A \rightarrow Up(A)]$ and $[A \rightarrow Down(A)]$)
- a *source relaxation/contraction* mechanism, if applied to $\mathcal{S}$ (i.e. $[S \rightarrow Up(S)]$ and $[S \rightarrow Down(S)]$).
  An example of such a mechanism for the case of taxonomy-based sources, founded on abduction, is described in [8].

These relative replacements can be used within context transition specifications. At the user interface level, this means that the user can have at his/her disposal two buttons, "Up" and "Down" for every component of the focal context. By pressing a button, the corresponding replacement is specified. Figure 1 sketches such a user interface. The option control labelled "Change" allows the user to indicate to the IM the component (S, q, or A) that should be changed in order to reach the new context. Then, it is the task of the IM to try to reach a context, using the approach indicated by the user, and subsequently to deliver this context to the user. A CTS defined by a relative replacement will be called *relative CTS*.

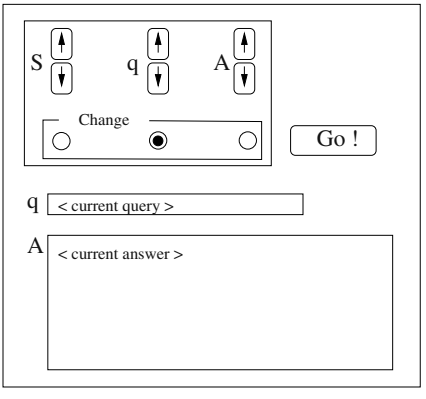


Fig. 1. A user interface for specifying context transitions through relative replacements

Below we enumerate all possible relative CTSs:

(1) $[q \rightarrow Up(q)/A]$: query relaxation resulting in answer enlargement
$[q \rightarrow Down(q)/A]$: query contraction resulting in answer reduction
(2) $[S \rightarrow Up(S)/A]$: source relaxation resulting in answer enlargement
$[S \rightarrow Down(S)/A]$: source contraction resulting in answer reduction
(3) $[A \rightarrow Up(A)/q]$: answer enlargement resulting in query relaxation
$[A \rightarrow Down(A)/q]$: answer reduction resulting in query contraction

oarieli@mta.ac.il

(4) $[S \rightarrow Up(S)/q]$: source relaxation resulting in query contraction
$[S \rightarrow Down(S)/q]$: source contraction resulting in query relaxation
(5) $[A \rightarrow Up(A)/S]$: answer enlargement resulting in source relaxation
$[A \rightarrow Down(A)/S]$: answer reduction resulting in source contraction
(6) $[q \rightarrow Up(q)/S]$: query relaxation resulting in source contraction
$[q \rightarrow Down(q)/S]$: query contraction resulting in source relaxation

6 Restricting the Relative CTSs

The three partial orders mentioned in Section 2 (and their interrelationships) can be exploited in order to restrict the set of relative CTSs to those for which the IM can indeed compute the target context. Below we describe how we can restrict relative replacements according to the CTS in which they appear.

Up/Down on Sources

As we saw earlier, source replacements appear in cases (2) and (4) of CTSs. In these cases, there is no need to restrict $Up(S)$ or $Down(S)$ because the IM can always find the target context. In particular, CTS (2) relies on query evaluation and CTS (4) on naming functions:

(2) $[S \rightarrow S'/A](c) = (S', q, S'(q))$, i.e. it relies on query evaluation
(4) $[S \rightarrow S'/q](c) = (S', n_{S'}(A), A)$, i.e. it relies on naming functions

Up/Down on Answers

Answer replacements appear in cases (3) and (5) of CTSs. CTS (2) relies on naming functions, while CTS (5) is still open:

(3) $[A \rightarrow A'/q](c) = (S, n_S(A'), A')$, i.e. it relies on naming functions
(5) $[A \rightarrow A'/S](c) = ?$, i.e. this is an open issue

Note that we can restrict $Up/Down(A)$ so that to support CTS (3) even if naming functions are not available (or if they are available, but there is no exact name for A'). This can be achieved by defining:

$$Up(A) = S(Up(q))$$
$$Down(A) = S(Down(q))$$

In this way, the IM can find the target context without using any naming function. i.e.

(3R) $[A \rightarrow S(Up(q))/q](c) = (S, Up(q), S(Up(q)))$
$[A \rightarrow S(Down(q))/q](c) = (S, Down(q), S(Down(q)))$

Another interesting remark is that by restricting $Up(A)$ or $Down(A)$ the IM can support CTS (5). This can be achieved by defining $Up(A)$ and $Down(A)$ as follows:

$$Up(A) = Up(S)(q)$$
$$Down(A) = Down(S)(q)$$

In this way, the IM can find the target context:

oarieli@mta.ac.il

(5R) $[A \rightarrow Up(S)(q))/S](c) = (Up(S), q, Up(S)(q))$
$[A \rightarrow Down(S)(q))/S](c) = (Down(S), q, Down(S)(q))$

Up/Down on Queries

Query replacements appear in cases (1) and (6) of CTSs:

(1) $[q \rightarrow q'/A](c) = (S, q', S(q'))$, i.e. relies on query evaluation
(6) $[q \rightarrow q'/S](c) = ?$, i.e. this is an open issue

CTS (1) relies on query evaluation, hence no restriction is needed. CTS (6) is still open but by restricting $Up/Down(q)$ the IM can support it, if naming functions are available. This can be achieved by defining $Up(q)$ and $Down(q)$ as follows:

$$Up(q) = n_{Down(S)}(A)$$
$$Down(q) = n_{Up(S)}(A)$$

In this way, the IM can find the target context:

(6R) $[q \rightarrow n_{Up(S)}(A)/S](c) = (Up(S), n_{Up(S)}(A), A)$
$[q \rightarrow n_{Down(S)}(A)/S](c) = (Down(S), n_{Down(S)}(A), A)$

In this section we showed how the IM can support all kinds of relative CTSs through restricted relative replacements.

7 Concluding Remarks

We introduced a unified formal framework which captures several interaction schemes between users and information systems. This unified view allowed us to describe more complex interaction mechanisms that those supported by the existing systems. The value of this unification is not only theoretical. It can be directly reflected in the interaction between the user and the information source, i.e in the user interface.

Further research includes specializing the introduced framework for the case where the source is a taxonomy-based source (such as a Web Catalog), a relational database, a semistructured database [1], or a Description-Logics database [5].

Acknowledgements. The first author wants to thank his wife for being an endless source of happiness and inspiration.

References

1. Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Series in Data Management Systems, 1999.

oarieli@mta.ac.il

2. Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *"Modern Information Retrieval"*. ACM Press, Addison-Wesley, 1999.
3. Alain Bidault, Christine Froidevaux, and Brigitte Safar. "Repairing Queries in a Mediator Approach". In *Proceedings of the ECAI'02*, Lyon, France, 2002.
4. Lee Dongwon. *"Query Relaxation for XML Model"*. PhD thesis, University of California, 2002.
5. F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. "Reasoning in Description Logics". In Gerhard Brewka, editor, *Principles of Knowledge Representation*, chapter 1, pages 191–236. CSLI Publications, 1996.
6. Terry Gaasterland. "Cooperative Answering through Controlled Query Relaxation". *IEEE Expert: Intelligent Systems and Their Applications*, 12(5), 1997.
7. Wen-Syan Li, K. Selçuk Candan, Quoc Vu, and Divyakant Agrawal. "Query Relaxation by Structure and Semantics for Retrieval of Logical Web Documents". *IEEE Transactions on Knowledge and Data Engineering*, 14(4), 2002.
8. Carlo Meghini and Yannis Tzitzikas. "An Abduction-based Method for Index Relaxation in Taxonomy-based Sources". In *Proceedings of MFCS 2003, 28th International Symposium on Mathematical Foundations of Computer Science*, number 2747 in Lecture notes in computer science, pages 592–601, Bratislava, Slovak Republic, August 2003. Springer Verlag.
9. E. Mena, V. Kashyap, A. Illarramendi, and A. Sheth. "Estimating Information Loss for Multi-ontology Based Query Processing". In *Proceedings of Second International and Interdisciplinary Workshop on Intelligent Information Integration*, Brighton Centre, Brighton, UK, August 1998.
10. Yannis Tzitzikas and Carlo Meghini. "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems". In *Seventh International Workshop on Cooperative Information Agents, CIA-2003*, number 2782 in Lecture Notes on Artificial Intelligence, pages 78–92, Helsinki, Finland, August 2003. (Best Paper Award).
11. Yannis Tzitzikas, Nicolas Spyratos, and Panos Constantopoulos. "Query Translation for Mediators over Ontology-based Information Sources". In *Second Hellenic Conference on Artificial Intelligence, SETN-2002*, Thessaloniki, Greece, April 2002.
12. Jeffrey D. Ullman. *"Principles of Database and Knowledge-Base Systems, Vol. I"*. Computer Science Press, 1988.

oarieli@mta.ac.il

Plan Databases: Model and Algebra

Fusun Yaman[1], Sibel Adali[2], Dana Nau[1], Maria L. Sapino[3], and V.S. Subrahmanian[1]

[1] University of Maryland, College Park MD, 20705, USA
{fusun,nau,vs}@cs.umd.edu
[2] Rensselaer Polytechnic Inst.,Troy, NY, 12180, USA
sibel@cs.rpi.edu
[3] Università di Torino, C.So Svizzera, 185-10149, Torino, Italy
mlsapino@di.unito.it

Abstract. Despite the fact that thousands of applications manipulate plans, there has been no work to date on managing large databases of plans. In this paper, we first propose a formal model of plan databases. We describe important notions of consistency and coherence for such databases. We then propose a set of operators similar to the relational algebra to query such databases of plans.

1 Introduction

Most of AI planning has focused on creating plans. However, the complementary problem of querying a collection of plans has not been studied. Querying plans is of fundamental importance in today's world. Shipping companies like UPS and DHL create plans for each package they have to transport. Multiple programs and humans need to query the resulting database of plans in order to determine how to allocate packages to specific drivers, to identify choke areas in the distribution network, and to determine which facilities to upgrade, etc. Likewise, every commercial port creates detailed plans to route ships into the port. Port officials need to determine which ships are on schedule, which ships are off schedule, which ships may collide with one another (given that one of them is off schedule) and so on. A similar application arises in the context of air traffic control - prior to takeoff, every flight has a designated flight plan and flight path. It is not uncommon for planes to be off schedule and/or off their assigned path. In other words, the plane may not be at the assigned location at the assigned time. Maintaining the integrity of air traffic corridors, especially in heavily congested areas (e.g. near Frankfurt or London airport), is a major challenge. Air traffic controllers need to be able to determine which flights are on a collision course, which flights are not maintaining adequate separation, which flights may intrude onto another flight's airspace and when.

These are just three simple applications where we need the ability to query collections of plans. *We emphasize that in this paper, we are not interested in creating plans, just in querying them.* The long version of this paper discusses

D. Seipel and J.M. Turull-Torres (Eds.): FoIKS 2004, LNCS 2942, pp. 302–319, 2004.
© Springer-Verlag Berlin Heidelberg 2004

oarieli@mta.ac.il

issues such as how to update databases of plans (which does involve some planning).

In this paper, we develop a formal model of a **plan database.** We then describe two important properties of such databases - **consistency and coherence** and present results that these properties are polynomially checkable. We then present a relational-algebra style **plan algebra** to query plan databases. In addition to the relational style operators, our algebra contains operators unique to plan databases.[1]

2 Plan Database Model

In this section, we introduce the basic model of a plan database. The concept of a plan is an adaptation of the notion of a plan in the well known PDDL planning language [4].

Definition 2.1. *A **planspace**, $\mathcal{PS}$, is a finite set of relations. A **planworld** pw over a planspace $\mathcal{PS}$ is a finite instance of each relation in $\mathcal{PS}$.*

We use the standard notion of a relational schema and domain of an attribute when describing planspaces. There are certain special relations called *numeric relations*.

Definition 2.2. *A **numeric relation** in a planspace is a relation $R(A_1, \ldots, A_n, V)$ where $\langle A_1, \ldots, A_n \rangle$ forms a primary key and V is of type real or integer.*

Note that a numeric relation $R(A_1, \ldots, A_n, V)$ represents a function f_R that maps $dom(A_1) \times \ldots \times dom(A_n) \rightarrow dom(V)$.

Example 2.1 (Package Example). A shipping company's planspace may use the following relations to describe truck locations, truck drivers, packages and valid routes:

- *at(object, location)*: specifies the location of drivers, trucks and package.,
- *route(location1, location2)*: specifies that there is a viable route from *location1* to *location2*.
- *in(package, truck)*: specifies information about which truck carries which package.
- *driving(driver, truck)*: specifies who is driving which truck.
- *fuel(truck, level)*: a numerical relation specifying the fuel level of each truck.

2.1 Actions

In this section, we define two types of actions, simple actions that occur instantaneously, and durative actions that take place over a period of time and

[1] Due to space constraints, we are unable to present operators to update plan databases.

oarieli@mta.ac.il

possibly require certain conditions to be true during this time. A *term* over A_i is either a member of $dom(A_i)$ or a variable over $dom(A_i)$. If $R(A_1, \ldots, A_n)$ is a relation in a planspace and t_i $(1 \leq i \leq n)$ is a term over A_i, then $R(t_1, \ldots, t_n)$ is an atom. Likewise, if A_i, A_j are attributes, then $A_i = A_j$ is an atom - if A_i, A_j are both attributes such that $dom(A_i), dom(A_j)$ are subsets of the reals, then $A_i \leq A_j, A_i < A_j, A_i \geq A_j$ and $A_i > A_j$ are atoms. If A is an atom, then A and $\neg A$ are literals.

If $R(A_1, \ldots, A_n, V)$ is a numeerical relation and δ is a real number, then $incr(R, \delta), decr(R, \delta)$ and $assign(R, \delta)$ are *nu-formulas*. These formulas say that the V column of R must be increased or decreased by (or set to) the value δ.

Definition 2.3. *A* ***simple action*** *w.r.t. a planspace $\mathcal{PS}$ is a 5-tuple consisting of*

- *Name: A string $\alpha(A_1, \ldots, A_n)$, where each A_i is a variable called a* parameter *of the action.*
- Precondition *$pre(\alpha)$: a conjunction of literals over the planspace.*
- Add list *$add(\alpha)$: set of atoms (denote what becomes true after executing the action).*
- Delete list *$del(\alpha)$: set of atoms (denote what becomes false after executing the action).*
- Numeric update list *$update(\alpha)$: set of nu-formulas.*

If $c_i \in dom(A_i)$ for $1 \leq i \leq n$, then $\alpha(c_1, \ldots, c_n)$ is an ***instance*** *of $\alpha(A_1, \ldots, A_n)$. As is standard practice in AI planning, we assume that all actions are* range restricted*, i.e., that all variables appearing in the condition and effects are parameters of the action. Thus, each action is unambiguously specified by its name.*

A simple action is executed instantaneously (in 0 time) and its effects are described by its add list, delete list and update list. We assume the nu-formulas in the numeric update list are executed in the following order: all *incr* updates first, then all *decr* updates, and then all *assign* updates.

Definition 2.4. *Suppose $\alpha(\boldsymbol{t})$ is an action instance. $\mathsf{L}(\alpha(\boldsymbol{t}))$ denotes the set of all numerical variables updated by $\alpha(\boldsymbol{t})$, $\mathsf{R}(\alpha(\boldsymbol{t}))$ denotes the set of numerical variables read by $\alpha(\boldsymbol{t})$ and $\mathsf{L}^*(\alpha(\boldsymbol{t}))$ is the set of all numerical variables whose value is being increased or decreased (but not assigned) by $\alpha(\boldsymbol{t})$.*

A simple action instance $\alpha(\boldsymbol{t})$ is *executable* in planworld *pw* if $pre(\alpha(\boldsymbol{t}))$ is satisfied by *pw*. The concept of mutual exclusion of actions specifies when two actions cannot co-occur.

Definition 2.5. *Two simple action instances $\alpha(\boldsymbol{t})$ and $\beta(\boldsymbol{z})$ are* ***mutually exclusive*** *if any of the following hold:*

$pre(\alpha(\boldsymbol{t})) \cap (add(\beta(\boldsymbol{z})) \cup del(\beta(\boldsymbol{z}))) \neq \emptyset$ $\quad pre(\beta(\boldsymbol{z})) \cap (add(\alpha(\boldsymbol{t})) \cup del(\alpha(\boldsymbol{t}))) \neq \emptyset$

$add(\alpha(\boldsymbol{t})) \cap del(\beta(\boldsymbol{z})) \neq \emptyset$ $\quad del(\alpha(\boldsymbol{t})) \cap add(\beta(\boldsymbol{z})) \neq \emptyset$

$\mathsf{L}(\alpha(\boldsymbol{t})) \cap \mathsf{R}(\beta(\boldsymbol{z})) \neq \emptyset$ $\quad \mathsf{L}(\beta(\boldsymbol{z})) \cap \mathsf{R}(\alpha(\boldsymbol{t})) \neq \emptyset$

$\mathsf{L}(\alpha(\boldsymbol{t})) \cap \mathsf{L}(\beta(\boldsymbol{z})) \neq \mathsf{L}^*(\alpha(\boldsymbol{t})) \cap \mathsf{L}^*(\beta(\boldsymbol{z}))$

oarieli@mta.ac.il

$\alpha(\boldsymbol{t})$ *and* $\beta(\boldsymbol{z})$ *are* mutually compatible *(i.e. they can occur at the same time) if they are not mutually exclusive.*

If A is an atom (resp. nu-formula, literal, conjunction of literals) and $w \in \{AtStart, AtEnd, OverAll\}$ then $w : A$ is an *annotated atom* (resp. nu-formula, literal, conjunction of literals).

Definition 2.6. *A* ***durative action*** *w.r.t. a planspace $\mathcal{PS}$ is a 5-tuple consisting of*

- Name: *this is the same as for simple actions.*
- Condition *cond*(α)*: set of annotated literals.*
- Add List *add*(α)*: set of $w : A$ where A is an atom and $w \in \{AtStart, AtEnd\}$.*
- Delete list *del*(α)*: set of $w : A$ where A is an atom and $w \in \{AtStart, AtEnd\}$.*
- Numeric update list *update*(α)*: set of $w : A$ where A is a nu-formula and $w \in \{AtStart, AtEnd\}$.*

Instances of durative actions are defined in the same way as for simple actions.

An ***action instance*** is an expression of the form $\alpha(\boldsymbol{t})$ for a vector $\boldsymbol{t}$ of the form $(c_1, \ldots, c_n)$ that assigns a constant to each variable in the name.

Example 2.2. We present a durative action, *load-truck(p,t,l)*, which loads package p into truck t at location l.

- *cond*(*load-truck(p,t, l)*) = {*atStart* : *at(p,l)*, *atStart* : *at(t,l)*, *overAll* : *at(t,l)*}. This says that when we start executing the *load-truck(p,t,l)* action, the truck and package should both be at location l and that throughout the execution of the *load-truck* action, the truck must be at location l (e.g. it cannot start moving during the loading operation).
- *add*(*load-truck(p,t,l)*) = {*atEnd* : *in-truck(p,t)*}. This says that the atom *in-truck*(p, t) is true at the end of the loading operation.
- *del*(*load-truck(p,t,l)*) = {*atStart* : *at(p,l)*}. Once we start executing the action, package p is no longer deemed to be at location l.
- *update*(*load-truck(p,t,l)*) = {}. There is no numeric update to be performed.

Other actions include *unload-truck* for unloading a package from truck, *board-truck* when a driver boards a truck, *disembark-truck* for disembarking the driver, and *walk* for displacing the driver on foot.

2.2 Plans and Plan Databases

In this section, we formally define a plan (and a plan database). Intuitively, a plan consists of a set of actions and constraints on the start and/or end times of actions. Due to space constraints, we use natural numbers to model time (formal calendars can be used with no difficulty). We use the variables $\mathsf{st}(\alpha(\boldsymbol{t}))$ and $\mathsf{et}(\alpha(\boldsymbol{t}))$, respectively, to denote the start and end times of an action.

oarieli@mta.ac.il

Note that any durative action $\alpha(\boldsymbol{t})$ can be split into three simple action instances. $\alpha_{start}(\boldsymbol{t})$ describes what happens at the start of the durative action. $\alpha_{interval}(\boldsymbol{t})$ is a simple action describing the conditions that must hold during the duration of the action (no changes occur during execution). $\alpha_{end}(\boldsymbol{t})$ is a simple action describing changes at the end. We use $SIMPLE(\alpha(\boldsymbol{t}))$ to denote this set of three simple actions associated with $\alpha(\boldsymbol{t})$. For $\alpha(\boldsymbol{t})$ to be executable, the precondition of $\alpha_{start}(\boldsymbol{t})$ must hold when we start executing α and the precondition of $\alpha_{end}(\boldsymbol{t})$ must hold at just before we finish executing. During execution, the precondition of $\alpha_{interval}(\boldsymbol{t})$ must be true. We now state this formally:

Definition 2.7. *Suppose pw_i denotes the planworld at time i. An action instance $\alpha(\boldsymbol{t})$ is **executable** in a sequence of planworlds $[pw_1, \ldots, pw_k]$ if all of the following hold:*

- $\alpha_{start}(\boldsymbol{t})$ *is executable in* pw_1*;*
- $\forall i, 1 < i < k$, $\alpha_{interval}(\boldsymbol{t})$ *is executable in* pw_i*;*
- $\alpha_{end}(\boldsymbol{t})$ *is executable in* pw_k*;*

where $k = \mathsf{et}(\alpha(\boldsymbol{t}))$

Furthermore action instances $\alpha(\boldsymbol{t})$ and $\beta(\boldsymbol{z})$ are *mutually exclusive* an action in $SIMPLE(\alpha(\boldsymbol{t}))$ and an action in $SIMPLE(\beta(\boldsymbol{z}))$ are mutually exclusive and happen at the same time.

Informally speaking, a plan is a set of action instances that are pairwise mutually compatible with each other and that are executed in accordance with some temporal constraints. We define execution constraints below.

Definition 2.8. *An **execution constraint** for an action $\alpha(\boldsymbol{t})$ is an expression of the form $\mathsf{st}(\alpha(\boldsymbol{t})) = c$ or $\mathsf{et}(\alpha(\boldsymbol{t})) = c$ where c is a natural number.*

Definition 2.9. *A **plan** w.r.t. a (finite) set $\mathcal{A}$ of actions is a pair $\langle \mathcal{A}', C \rangle$ where $\mathcal{A}'$ is a set of action instances from $\mathcal{A}$ and C is a set of execution constraints w.r.t. actions in $\mathcal{A}'$.*[2]

A plan is definite *if for all actions in $\mathcal{A}'$, there are two execution constraints in C, one for constraining the start time and the other constraining the end time.*

Goals. In AI planning, a plan normally is generated to achieve some *goal* that is represented as a set of literals g. Though we do not define goals explicitly, there is no loss of generality because each goal g can be encoded in the plan as a special action whose pre-condition is g and whose effects are empty. The duration of this action specifies how long the goal conditions should be protected in the plan world after the completion of the plan.

We only consider definite plans in this paper, rather than allowing the start and end times of actions to vary.

[2] A minor problem here is how to handle plans in which the same action (e.g., *refuel(truck1)*) occurs more than once. An easy way to ensure that distinct action instances have a different names it to give each action instance an additional parameter called an *action identifier* that is different for each distinct action instance, e.g., *refuel(truck1,instance01)* and *refuel(truck1,instance02)*.

oarieli@mta.ac.il

Example 2.3. Suppose we have packages p_1, p_2 at locations l_1, l_2 respectively. We have one truck t_1 at l_1 and a driver d in it. We want to deliver p_1 and p_2 to l_3. One possibility is to load p_1, then pick up p_2, and then go to the destinations. Here, $\langle \mathcal{A}', C \rangle$ is:

- $\mathcal{A}' = \{ a_1 = \textit{load-truck}(p_1, t_1, l_1),\ a_2 = \textit{drive-truck}(t_1, l_1, l_2, d),\ a_3 = \textit{load-truck}(p_2, t_1, l_2),\ a_4 = \textit{drive-truck}(t_1, l_2, l_3, d),\ a_5 = \textit{unload-truck}(p_1, t_1, l_3),\ a_6 = \textit{unload-truck}(p_2, t_1, l_3)\}$
- $C = \{st(a_1) = 1, et(a_1) = 2, st(a_2) = 3, et(a_2) = 5, st(a_3) = 6, et(a_3) = 7, st(a_4) = 8, et(a_4) = 12, st(a_4) = 13, et(a_4) = 14, st(a_4) = 13, et(a_4) = 14\}$

C indicates an intuitive order for a package: load, drive, unload. Notice that two unload operations are performed concurrently.

Note that each action in a plan is an abstract realization of a physical process. For example, the action $\textit{drive}(t_1, l_1, l_2, d)$ is a syntactic representation of the physical action that a driver performs of driving from one place to another. Due to exogenous events, an action may not always succeed when carried out in the real world.

Definition 2.10. *A **plan database** is a 4-tuple $\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$, where $\mathcal{PS}$ is a planspace, pw is the current planworld,* `plans` *is a finite set of plans and* `now` *is the current time.*

3 Consistency and Coherence of Plan DBs

Not all plan databases are consistent. For example, if we have only 50 gallons of fuel at a given location at some time T, and two different plans each plan to use 40 of those 50 gallons at that location at time T, then we would have an inconsistency. Coherence, on the other hand, intuitively requires that the plans be executable: all plans in the database must, for example, have preconditions that are valid w.r.t. the other plans in the database and the initial planworld. To formalize these notions, we first introduce the concept of future planworlds.

3.1 Future Planworlds

Throughout this section, we let $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ be some arbitrary but fixed plan database. We use $S_{plans}(i)$ (resp. $E_{plans}(i)$) to denote the set of all actions in `plans` whose start (resp. end) time is i. $I_{plans}(i)$ is the set of the actions that start before time i and end after time i. The set of ***active actions*** at time i w.r.t. a given set *plans* of plans is defined as:

$$Active_{plans}(i) = \left(\bigcup_{\alpha \in S_{plans}(i)} \alpha_{start}\right) \cup \left(\bigcup_{\alpha \in E_{plans}(i)} \alpha_{end}\right) \cup \left(\bigcup_{\alpha \in I_{plans}(i)} \alpha_{interval}\right).$$

Suppose `plans` is given, the current time is i, and the plan world at time i is pw_i. What should the planworld at time t_{i+1} be, according to `plans`? We use $\mathcal{PW}^i_{plans}(R)$ to denote the extent of relation R at time i w.r.t. `plans`.

oarieli@mta.ac.il

Definition 3.1 (future planworlds). *For all R,*

$$\mathcal{PW}^0_{plans}(R) = R$$
$$\mathcal{PW}^{i+1}_{plans}(R) = (\mathcal{PW}^i_{plans}(R) - Del_R(i, \texttt{plans})) \bigcup Add_R(i, \texttt{plans}),$$

where $Add_R(i, \texttt{plans})$ and $Del_R(i, \texttt{plans})$ are the set of all insertions and deletions, respectively to/from relation R by all actions in $S_{\texttt{plans}}(i) \cup E_{\texttt{plans}}(i)$. In the special case where R is a numeric relation, all tuples whose values are updated will be in $Del_R(i, \texttt{plans})$ and $Add_R(i, \texttt{plans})$ will contain the updated tuples with the new values, v'. If a numeric variable is updated at time i by a set of concurrent plan updates, then its new value will be computed as the old value plus the sum of all increases and decreases.

Assumptions. The above definition assumes that (i) when an action is successfully executed, its effects are incorporated into the planworld one time unit after the action's completion; (ii) all the actions in `plans` are successfully executable until information to the contrary becomes available; (iii) none of the actions are mutually exclusive with each other.

We now formally define the concept of consistency. Intuitively, consistency of a plan database requires that at all time points t, no two actions are mutually exclusive.

Definition 3.2. *Let P be a set of plans,* now *be the current time and e be the latest ending time in P. P is **consistent** if for every t,* now $\leq t \leq e$, *$Active_P(t)$ does not contain any two simple actions that are mutually exclusive.*

The following algorithm can be used to check consistency of a set P of plans.

Algorithm *ConsistentPlans*$(P,$ now$)$
  $L =$ ordered time points either at or one time unit before an action starts or ends in P;
  while L is not empty **do**
    $t =$ First member of L; $L = L$ - $\{t\}$;
    if $(\exists \alpha, \beta \in Active_P(t))$ α and β are mutually exclusive
      then return $false$;
  return *true*.

The reader can verify that the loop in this algorithm is executed at most $4n$ times, where n is the number of actions in P. Note that consistency of a plan database does not mean that the plan can be executed. To execute all the plans in a plan database, we need to ensure that the precondition of each action is true in the state (i.e. at the time) in which we want to execute it. The notion of coherence intuitively captures this concept.

Definition 3.3. *Suppose pw is the planworld at time* now *and P is a consistent set of plans. Suppose e is the latest ending time of any action in P. P is **coherent** iff for every* now $\leq t \leq e$ *every simple action in $Active_P(t)$ is executable in $\bigcup_R \mathcal{PW}^t_P(R)$ where $pw = \bigcup_R \mathcal{PW}^{\text{now}}_P(R)$.*

Clearly, we would always like a plan database to be both consistent (no conflicts) and coherent (executable). The following algorithm may be used to check for coherence.

oarieli@mta.ac.il

Algorithm $CoherentPlans(P, \mathsf{now}, pw)$
L = ordered time points either at or one time unit before an action starts or ends in P;
while L is not empty **do**
t = First member of L; $L = L$ - $\{t\}$;
if $(\exists \alpha \in Active_P(t)) pw \not\models pre(\alpha)$ **then return** $false$;
if $(\exists \alpha, \beta \in Active_P(t))$ α and β are mutually exclusive
then return $false$;
$pw = (pw - \bigcup_R Del_R(t, P)) \bigcup (\bigcup_R Add_R(t, P))$;
return $true$.

Goals. In AI planning, a plan normally is generated to achieve some *goal* that is represented as a set of lierals g. Though we do not define goals explicitly, there is no loss of generality because each goal g can be encoded in the plan as a special action whose pre-condition is g and whose effects are empty. The duration of this action specifies how long the goal conditions should be protected in the plan world after the completion of the plan.

Suppose we already know a given plan database is consistent (coherent), and we want to modify the set of plans in the plan database (but not the other components of the plan DB). The following two theorems provide sufficient conditions to check if the modified set of plans is consistent (coherent).

Theorem 3.1. *Suppose a plan database* $PLDB = \langle \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now} \rangle$ *is consistent. Let* $PLDB' = \langle \mathcal{PS}, pw, \mathtt{plans}', \mathsf{now} \rangle$. $PLDB'$ *is consistent if*

- $Actions(\mathtt{plans}') \subseteq Actions(\mathtt{plans})$ *and*
- $Constraints(\mathtt{plans}') \subseteq Constraints(\mathtt{plans})$,

where $Actions(\mathtt{plans}')$ *is the set of all actions in all plans in* $\mathtt{plans}'$ *and* $Constraints(\mathtt{plans}')$ *is the set of all constraints in all plans in* $\mathtt{plans}'$.

Theorem 3.2. *We use the same notation as in theorem 1. Suppose a plan database* $PLDB = \langle \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now} \rangle$ *is coherent and* $\mathtt{plans}'$ *satisfies the conditions in Theorem 1.* $PLDB'$ *is coherent if:*

1. $Cond(\mathtt{plans}') \cap Effects(\mathtt{plans} - \mathtt{plans}') \equiv \emptyset$, *or*
2. *All actions in* $\mathtt{plans}'$ *end before any action in* $\mathtt{plans} - \mathtt{plans}'$ *starts.*

Here $Cond(P)$ *is the set of preconditions of all actions in* P *and* $Effects(P)$ *is the set of all the effects of all actions in* P.

4 Plan Database Algebra

We now define a plan database algebra (PDA for short) to query plan databases. PDA contains selection, projection, union, intersection, and difference operators. In addition, we introduce a *coherent selection* operator cs and a *coherent projection* operator cp which is used to ensure coherence properties. A new ***fast forward*** operator can be used to query the database about future states. Note

oarieli@mta.ac.il

that this is different from a temporal database where future temporal states are explicitly represented. In a plan database, all we are given explicitly is that various actions are scheduled to occur at various times, and need to reason about when these actions are performed and their effects in order to answer queries about future states. Just reading the database is not adequate.

4.1 Future Plan Databases

In order to achieve this goal, we first define the concept of *future plan databases*. Recall that in Section 3.1, we introduced "future planworlds". This definition assumed that the plan database was coherent. However, this may not always be the case. Future plan databases describe the state of the plan database by *projecting* into the future. We assume we start with a consistent (but not necessarily coherent) database.

Definition 4.1. *Suppose* $\langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle$ *is a plan database and that the current time is* now*. The* ***future plan database*** $PossDB$ *at time* i *for* $i \geq \mathsf{now}$ *is defined inductively as follows:*

1. *For* $i = \mathsf{now}$*:* $PossDB^i(\langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle) = \langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle$ *and* $\texttt{plans}^i = \texttt{plans}$.
2. *For* $i > \mathsf{now}$*: Suppose* $PossDB^{i-1}(\langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle) = \langle \mathcal{PS}, pw^{i-1}, \texttt{plans}^{i-1}, (i-1) \rangle$. *Then* $PossDB^i(\langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle) = \langle \mathcal{PS}, pw^i, \texttt{plans}^i, i \rangle$*, where:*
   a) $pw^i = (pw^{i-1} - \bigcup_R Del_R(i-1, \texttt{plans}^{i-1})) \bigcup (\bigcup_R Add_R(i-1, \texttt{plans}^{i-1}))$.
   b) $\texttt{plans}^i = \{\langle A, C \rangle \mid \langle A, C \rangle \in \texttt{plans}^{i-1}, (A \cap CannotStart) = \emptyset\}\}$.
   c) $CannotStart = \{\alpha \mid \alpha_{sub} \in Active, sub \in \{end, start, interval\}, pw^i \not\models pre(\alpha_{sub})\}$.
   d) $Active = Active_{plans^{i-1}}(i)$

The above definition inductively defines the plan database at time i by constructing it from the plan database at time $(i-1)$.

4.2 Selection Conditions

Before defining selection, we first need to define selection conditions. Suppose $\mathcal{PS}$ is some arbitrary but fixed planspace. As usual, we assume the existence of variables over domains of all attributes in the relations present in the planspace. In addition, we assume the existence of a set of variables $Z_1, Z_2, \ldots$ ranging over plans, a set $A_1, A_2, \ldots$ of variables ranging over actions, and a set $Y_1, Y_2, \ldots$ of variables ranging over tuples (of a planworld). A *plan term* is either a variable of any of the above three kinds, a constant of the appropriate kind, or of the form $V.a$ where V is a variable of the above kinds and a is an attribute of the term denoted by V. If $V_1, \ldots, V_k$ are all plan terms then $\langle V_1, \ldots, V_k \rangle$ is a plan term denoting a tuple. Terms denoting actions and plans have special attributes `START` and `END` that correspond to the start and end time of actions and plans. In addition, terms denoting actions have a special `name` attribute. In the following definition, we assume the existence of some arbitrary but fixed planspace.

oarieli@mta.ac.il

Definition 4.2. *Atomic selection conditions (ASCs) are inductively defined as follows:*

1. *If Y is a tuple term and R is a relation, then $Y \in R$ is an ASC.*
2. *If P is a plan term and A is an action term, then $A \in P$ is an ASC.*
3. *If t_1, t_2 are terms of the same type, then $pt_1 = pt_2$ is an ASC.*
4. *If t_1, t_2 are terms of the same type and the type has an associated linear ordering $\leq$, then $t_1 \ \underline{op}\ t_2$ is an ARC, where $\underline{op} \in \{\leq, <, >, \geq, <>\}$.*

Definition 4.3. *A* simple *plan database condition (simple PDC) is inductively defined as: (i) every ASC is a simple PDC, and (ii) if X_1, X_2 are simple PDCs, then so are $(X_1 \wedge X_2)$ and $(X_1 \vee X_2)$.*

If X is a simple plan database condition and I is either a variable (over integers) or an integer, then the expression $[I] : X$ is a plan database condition (PDC).

The condition $[I] : X$ holds if the condition X evaluates to true at time I in the underlying plan database. If I is a constant, then the PDC is called a *time bounded expression*. Otherwise, we say that the PDC is an *unbounded expression*.

Definition 4.4 (satisfaction). Let X be a ground simple PDC and $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ be a plan database. The satisfaction of all atomic selection conditions by a $PLDB$ is defined in the obvious way. In addition, if p is a plan term, then $PLDB$ satisfies $p.\texttt{END}\ \underline{op}\ c$ if for all actions $\alpha \in p$, $\mathsf{et}(\alpha) \leq \texttt{now}$, $z = max\{e \mid \mathsf{et}(\alpha) = e, \alpha \in p\}$ and $z\ \underline{op}\ c$ holds. Similarly, $PLDB$ satisfies $p.\texttt{START}\ \underline{op}\ c$ if there is an action $\alpha \in p$ such that $\mathsf{st}(\alpha) < \texttt{now}$, $z = min\{s \mid \mathsf{st}(\alpha) = s, s \leq \texttt{now}, \alpha \in p\}$ and $z\ \underline{op}\ c$ holds. If α is an action term and p is a plan, then $PLDB$ satisfies $\alpha \in p$ if $p \in \texttt{plans}$, $p = \langle \mathcal{A}', \mathcal{C} \rangle$ and $\alpha \in \mathcal{A}'$. If a is an action term in plan p then $PLDB$ satisfies $\alpha.\texttt{START}\ \underline{op}\ c$ iff $\mathsf{st}(\alpha) = s$, $s \leq \texttt{now}$ and $s\ \underline{op}\ c$ holds. Similarly, $PLDB$ satisfies $\alpha.\texttt{END}\ \underline{op}\ c$ iff $\mathsf{et}(\alpha) = e, e < \texttt{now}$ and $e\ \underline{op}\ c$ holds. If R is a relation name then $PLDB$ satisfies $Y \in R$ succeeds for a tuple term Y iff tuple Y is in the relation R according to pw. Suppose i is an integer, $PLDB$ satisfies $[I] : X$ if and only if X is true in $PossDB^i(PLDB)$. If $[I] : X$ is a non-ground PDC then, $PLDB$ satisfies $[I] : X$ if there exists a ground instance $[I] : X\gamma$ such that $PLDB \models [I] : X\gamma$. If $[I] : X$ is an unbounded (i.e. I is a variable) PDC then $PLDB$ satisfies $[I] : X$ iff there exists an integer i such that $PLDB \models [I] : X$.

As usual, we use the symbol $\models$ to denote satisfaction.

Example 4.1. To find all actions that finish before time 20, we can write $(A.\texttt{END} \leq 20)$. To find all plans that will finish successfully, we can write $[I] : (Z.\texttt{END} \leq I)$. In this expression, we want to find a time instance I where all plans in the database at time I finish successfully (before time I).

The following algorithm finds all plans that successfully end before time i. The algorithm is useful if a plan database is not coherent. If the plan database is coherent, we know for certain that all the plans in the plan database will succeed unless an exogenous real world event intervenes (which would lead to a database update).

oarieli@mta.ac.il

Algorithm PlansSuccessfullyEnd(PDB, i)
$Ans = \emptyset$;
$\langle \mathcal{PS}, pw^i, \texttt{plans}^i, i \rangle = PossDB^i(PDB)$;
while $\texttt{plans}^i \neq \emptyset$ **do**
Select $\langle \mathcal{A}, \mathcal{C} \rangle \in \texttt{plans}^i$; $\texttt{plans}^i = \texttt{plans}^i - \{\langle \mathcal{A}, \mathcal{C} \rangle\}$;
if there is no $\alpha \in \mathcal{A}$ such that $\mathsf{et}(\alpha) > i$ **then** $Ans = Ans \cup \{\langle \mathcal{A}, \mathcal{C} \rangle\}$
return Ans.

It is easy to see that that above algorithm can be executed in time proportional to the number of plans in the plan database.

4.3 Selection

The selection operation finds all plans (and their associated information) that satisfy a specific condition.

It is important to note that selection may not preserve coherence. For instance, suppose we have a database containing five plans $p_1, \ldots, p_5$ and suppose p_1, p_2, p_3 satisfy the selection condition. Then these are the plans that the user wants selected. However, p_2 may have actions in it that depend upon the prior execution of p_4 (otherwise the preconditions of p_2 may not be true). Coherence would require that we add p_4 to the answer as well. For this reason, we define two versions of the selection operator - ordinary selection which does not necessarily guarantee coherence, and coherent selection which would add a minimal number of extra plans to guarantee coherence.

Definition 4.5. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a plan variable Z. The *plan selection* operation, denoted by $\sigma_{[I]:X}\ PLDB(Z) = \langle \mathcal{PS}, pw, \texttt{plans}', \mathsf{now} \rangle$, is computed as $\texttt{plans}' = \{\langle \mathcal{A}, \mathcal{C} \rangle \mid \langle \mathcal{A}, \mathcal{C} \rangle \in sol(Z)\}$, where

$$sol(Z) = \{\langle \mathcal{A}, \mathcal{C} \rangle \in \texttt{plans} \mid PLDB \models [I] : X/\{Z = \langle \mathcal{A}, \mathcal{C} \rangle\} \text{ and } \nexists I' < I \text{ such that } PLDB \models [I'] : X/\{Z = \langle \mathcal{A}, \mathcal{C} \rangle\}\}.$$

Proposition 4.1. *If $PLDB$ is consistent, then according to Theorem 1, $\sigma_{[I]:X}\ PLDB(Z)$ is also consistent. If $PLDB$ is coherent and $\sigma_{[I]:X}\ PLDB(Z)$ satisfies either of the conditions in Theorem 2, then $\sigma_{[I]:X}\ PLDB(Z)$ is also coherent.*

Example 4.2. Suppose we want to retrieve all plans in which a certain driver, say Paul, drives the truck. We can write the following plan selection query: $\sigma_{[I]:X}\ PLDB(Z)$ where $X = (A = \textit{drive-truck}(\_, \_, \_, \textit{paul}) \wedge A \in Z)$.

Suppose the initial plan database $PLDB$, contains the following plans;

- $P_1 = \langle \{a_1 = \textit{board-truck}(\textit{paul}, t_1, c_1), a_2 = \textit{board-truck}(\textit{paul}, t_1, c_2), a_3 = \textit{board-truck}(\textit{ted}, t_2, c_3)\}, \{\mathsf{st}(a_1) = 1, \mathsf{et}(a_1) = 3, \mathsf{st}(a_2) = 9, \mathsf{et}(a_2) = 11, \mathsf{st}(a_3) = 1, \mathsf{et}(a_3) = 3\}\rangle$
- $P_2 = \langle \{a_4 = \textit{drive-truck}(t_1, c_1, c_2, \textit{paul}), a_5 = \textit{drive-truck}(t_2, c_1, c_2, \textit{ted})\}, \{\mathsf{st}(a_4) = 4, \mathsf{et}(a_4) = 8, \mathsf{st}(a_5) = 6, \mathsf{et}(a_2) = 11\}\rangle$
- $P_3 = \langle \{a_6 = \textit{walk}(\textit{paul}, c_2, c_3)\}, \{\mathsf{st}(a_6) = 12, \mathsf{et}(a_6) = 16\}\rangle$

oarieli@mta.ac.il

and the current time is 0. In this case, the above query returns only P_2. However the plan database which contains just P_2 is not coherent at time 0 because at time 4, Paul will not be in truck t_1 which is one of the conditions of action a_4. The coherent selection operation will fix this.

Example 4.3. Suppose we want to retrieve all plans in which the same driver has to deliver items to at least three different places. We can write the following plan selection expression: $\sigma_{[I]:X}\, PLDB(Z)$ where $X = (A1 = \mathit{drive\text{-}truck}(\_, \_, L1, D) \wedge A2 = \mathit{drive-truck}(\_, \_, L2, D) \wedge A3 = \mathit{drive-truck}(\_, \_, L3, D) \wedge A1 \in Z \wedge A2 \in Z, \wedge A3 \in Z \wedge L1 \neq L2 \wedge L1 \neq L3 \wedge L2 \neq L3)$.

4.4 Coherent Selection

Selection is guaranteed to preserve consistency, but not coherence. Fortunately, we can restore coherence by using the algorithm `ClosePlans` below. The algorithm invokes a subroutine called `SupportivePlans`(P, F, t). For every action $\alpha \in F$, `SupportivePlans` nondeterministically[3] selects a plan in P that contains an action β with an effect e which establishes the precondition of α. It also ensures that $\mathsf{st}(\beta)$ (resp. $\mathsf{et}(\beta)$) is less than t, if e is an effect of β_{start} (resp. β_{end}). `SupportivePlans` returns the set of selected plans. The algorithm is guaranteed to terminate if the input plan DB `plans` is coherent wrt pw and `now`.

Algorithm `ClosePlans`$(\mathcal{PS}, pw, \mathtt{plans}, \mathsf{now}, \mathtt{plans}')$
  $last$ = Latest ending time in `plans'`;
  $t = \mathsf{now}$; $pw_t = pw$;
  while $t \leq last$ do
    $A \equiv Active_{plans'}(t)$
    if $A \equiv \emptyset$ then
      $pw_{t+1} = pw_t$;t= t+1;
    else if $\forall \alpha \in A, pw_t \models pre(\alpha)$ **then**
      $pw_{t+1} = pw_t - Del_{plans'}(t) + Add_{plans'}(t)$;
      $t = t + 1$;
    else
      $F \equiv \{\alpha | \alpha \in A, pw_t \not\models pre(\alpha)\}$;
      $P =$ `SupportivePlans`$(\mathtt{plans} - \mathtt{plans}', F, t)$;
      $t =$ Earliest start time of actions in P;
      $\mathtt{plans}' \equiv \mathtt{plans}' \cup P$
      $last$ = Latest ending time in `plans'`;
  return $\mathtt{plans}'$

We now define the coherent selection operator cs that guarantees coherence.

Definition 4.6. Suppose $PLDB = \langle \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a plan variable Z. The ***coherent selection*** operation, denoted by $\mathsf{cs}_{[I]:X}\, PLDB(Z) = \langle \mathcal{PS}, pw, \mathtt{plans}*, \mathsf{now} \rangle$, is given by:

[3] Note that any nondeterministic operation can be made deterministic by defining a linear order on all choices and simply choosing the choice that is minimal w.r.t. the linear order. Due to space limitations, we do not pursue this option here.

oarieli@mta.ac.il

- $\langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle = \sigma_{[I]:X}\ PLDB(Z)$
- $\texttt{plans} *= \texttt{ClosePlans}(\mathcal{PS}, pw, \texttt{plans}, \texttt{now}, \texttt{plans}')$

Example 4.4. Let us return to Example 4.2, where we want to select all plans in which Paul drives. The *coherent selection* operation would return the plan DB containing both P_2 and P_1 which will be coherent.

4.5 Projection

The projection operation selects plans which contain actions that satisfy a specific condition. For a plan, only the actions that satisfy the conditions are kept, the others are removed from the plan. As in the case of *selection*, the coherence property may be violated after a projection. Later, we will introduce a coherence preserving projection operation that establishes coherence by reinserting some actions and/or plans removed during projection to reestablish the necessary coherence property.

Definition 4.7. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a variable A denoting an action. The ***action projection*** operation, denoted $\Pi_C\ PLDB(A) = \langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle$, is defined as:

- $\texttt{plans}' = \{\langle \mathcal{A}^*, \mathcal{C}^* \rangle \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}, \mathcal{A}^* = \{\alpha \mid \alpha \in \mathcal{A}' \text{ and } \alpha \in sol(A)\}, \mathcal{C}^* = rest(\mathcal{C}, \mathcal{A}^*)\}$,

where

- $sol(A) = \{\alpha \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}$ and $PLDB \models [I] : X/\{A = \alpha\}$ and $\nexists I' < I$ such that $PLDB \models [I'] : X/\{A = \alpha\}\} \cup \{\alpha \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}$ and $\mathsf{st}(\alpha) \leq \texttt{now}\}$;
- $rest(\mathcal{C}, \mathcal{A}^*)\} = \bigcup_{\alpha \in \mathcal{A}^*} \{\text{all execution constraints for } \alpha \text{ in } \mathcal{C}\}$.

We note that the action projection will return actions that satisfy the given conditions and started already.

Proposition 4.2. *If $PLDB$ is consistent then so is $\Pi_{[I]:X} PLDB(A)$. If $PLDB$ is coherent and $\Pi_{[I]:X}\ PLDB(A)$, satisfies either of the conditions in Theorem 2, then $\Pi_{[I]:X}\ PLDB(A)$, is also coherent.*

Example 4.5. Suppose we want to retrieve plans only consisting of *drive-truck* actions. Specifically, we only want to keep those actions for which there exists, in their own plan, another delivery that has the same driver, and the second delivery happens after x time units. We can use the following plan projection query: $\Pi_{[I]:X}\ PLDB(A)$ where $X = (A1 = drive\text{-}truck(\_, \_, \_, D) \wedge A2 = \text{drive-truck}(\_, \_, \_, D) \wedge\ A1 \neq A2 \wedge et(A1) - et(A2) = x, \wedge(A = A1 \vee A = A2))$.

Example 4.6. Consider once more the selection query in Example 4.4. A projection operation with the same condition on the same plan database will yield a plan database with the following single plan in it: $P_2' = \langle \{a_4 = drive(t_1, c_1, c_2, paul)\}$,
$\{\mathsf{st}(a_4) = 4, \mathsf{et}(a_4) = 8\}\rangle$. As explained earlier, a_4 will fail because there is no driver in $truck_1$.

oarieli@mta.ac.il

4.6 Coherent Projection

We now define a ***closed plan projection*** operator cp similar to the coherent selection operator. It will return the plans with actions that satisfy the selection criteria as well as the other actions needed to make the projected set of plans coherent.

Definition 4.8. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PRC. The ***closed plan projection operation***, denoted $\texttt{cp}_{[I]:X}\ PLDB(A) = \langle \mathcal{PS}, pw, \texttt{plans}*, \texttt{now} \rangle$, is given by:

- $\langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle = \pi_{[I]:X}\ PLDB(A, Z)$
- $\texttt{plans}* = \texttt{CloseActions}(\mathcal{PS}, pw, \texttt{plans}, \texttt{now}, \texttt{plans}')$

The definition of closed projection requires a `CloseActions` procedure which is a slight variation of the `ClosePlans` algorithm. Instead of calling `SupportivePlans`, it calls a `SupportiveActions` which is a slight variant of `SupportivePlans`: basically this procedure returns plans restricted to the supporting actions. The following example shows the use of the coherent projection operator.

Example 4.7. Let us return to the case of Example 4.6 and use coherent projection instead of projection. The resulting plan DB contains two plans:

- $P_1' = \langle \{a_1 = \textit{board-truck}(paul, t_1, c_1)\}, \{\texttt{st}(a_1) = 1, \texttt{et}(a_1) = 3\} \rangle$,
- $P_2' = \langle \{a_4 = \textit{drive-truck}(t_1, c_1, c_2, paul)\}, \{\texttt{st}(a_4) = 4, \texttt{et}(a_4) = 8\} \rangle$.

This database is coherent. Notice the difference between number of actions added by coherent selection and coherent projection. In the first case, the total number of actions added into the plan database is three whereas in the second case it is only one. This is because coherent selection includes a plan with all its actions, whereas coherent projection only includes the necessary actions.

4.7 Fast Forward

In this section we define the ***fast-forward*** operator which returns future states of a plan database that satisfy various PDC conditions. The fast forward operation can be thought of as a projection operation into the future. Note however, that unlike a temporal database, we cannot look just at the relational state - we must also see how this relational state changes over time as the various actions in the plan database are executed according to the given schedule.

Definition 4.9. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC. The ***fast forward*** of database $PLDB$ with respect to $[I] : X$, is $\Gamma_{[I]:X}(PLDB) = PossDB^I(PLDB)$, where I is the smallest integer such that $PLDB \models [I] : X$ if such an I exists. If no such I exists, then $\Gamma_{[I]:X}(PLDB)$ is undefined.

Proposition 4.3. *If $PLDB$ is consistent/coherent and $\Gamma_{[I]:X}(PLDB)$ is defined then $\Gamma_{[I]:X}(PLDB)$ is also consistent/coherent.*

oarieli@mta.ac.il

4.8 Union, Intersection, Difference

In this section we describe the *union, difference* and *intersection* operations for plan databases. We first define the notion of *union compatibility* which simply states that the data in two plan worlds must have same values for the same numeric variables. The reason for this is that if one plan world says there 10 gallons of fuel, and another plan world says there are 20, then the union yields something claiming there are both 10 and 20 gallons of fuel which is problematic. Unlike intersection and difference, union does not necessarily preserve consistency even when the plan databases involved are union compatible. However, in Theorem 3 below, we state some conditions that are sufficient to preserve consistency. Two databases $\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ and $\langle \mathcal{PS}, pw', \texttt{plans}', \texttt{now} \rangle$ are *union compatible* if every numeric variable f that is both in pw and pw' has the same value in both plan worlds.

Definition 4.10. Let $PLDB_1 = \langle \mathcal{PS}, pw_1, \texttt{plans}_1, \texttt{now} \rangle$ and $PLDB_2 = \langle \mathcal{PS}, pw_2, \texttt{plans}_2, \texttt{now} \rangle$ be two union compatible plan databases. Suppose the plans in $\texttt{plans}_2$ are renamed so that there are no plans with the same identifier in both databases. Then, the ***union*** of $PLDB_1, PLDB_2$, denoted $PLDB_1 \cup PLDB_2$ is given by

$$PLDB_1 \cup PLDB_2 = \langle \mathcal{PS}, pw_1 \cup pw_2, \texttt{plans}_1 \cup \texttt{plans}_2, \texttt{now} \rangle.$$

The following theorem states conditions guaranteeing consistency of the union of two union-compatible plan databases.

Theorem 4.1. *Suppose $PLDB_1$ and $PLDB_2$ are consistent. $PLDB_1 \cup PLDB_2$ is consistent if $\forall(\alpha \in Actions(\texttt{plans}_1), \beta \in Actions(\texttt{plans}_2))$ either of the following holds:*

1. $(Cond(\alpha) \cup Effects(\alpha)) \bigcap (Cond(\beta) \cup Effects(\beta)) = \emptyset$;
2. $\mathsf{st}(\alpha) > \mathsf{et}(\beta)$ *or* $\mathsf{st}(\beta) > \mathsf{et}(\alpha)$;

where $Actions(\texttt{plans})$ is the set of all actions of all plans in $\texttt{plans}$, *$Cond(\alpha)$ is the set of conditions of α and $Effects(\alpha)$ is the set of all effects of α.*

Theorem 3 is intuitive. If any two actions that access the same atoms do not overlap in time, then they cannot be mutually exclusive because none of their simple actions will happen at the same time. Any other two actions with overlapping executions will not be mutually exclusive since they don't modify the truth values of the same atoms.

The intersection and difference between two plan databases can be defined analogously, but we omit the definitions due to lack of space. Note that union, intersection, and difference may not be coherent even if the input plan DBs are coherent. We can define *coherent* union, intersection and difference operators in a manner similar to the coherent selection and projection operators.

oarieli@mta.ac.il

5 Related Work

To date, there has been no other work on developing plan databases. We are aware of no formal query language for querying plans analogous to the relational algebra or relational calculus. However, there are two related areas: case based planning and temporal databases.

The goal of case based planning [6] is to store plans in a "case base" so that when we need to solve a new planning problem, we can examine the case base and identify similar plans that can be modified to solve the new planning problem. Our goal in this paper is very different. We are interested in *querying large databases of plans* so that different applications can perform their tasks. Such applications involve logistics where a transportation company may wish to examine plans and schedules to determine how to allocate resources (using operations research methods perhaps) as well as to analyze traffic, as well as air traffic control where we wish to identify when and where aircraft will be in the future so as to avoid potential mishaps. Some important aspects of our framework and consistency and coherence of the database. In contrast, case based planners do not require consistency nor coherence because the case base is not a set of plans being executed; rather, it is a library and the queries to this library concentrate on similarity issues.

There are also connections between our work and work in temporal databases [9,2]. In temporal relational databases, we have two kinds of time: transaction time and valid time. Transaction time databases store information about when a given tuple was inserted into a relation, when updates were made, etc. Therefore, such databases deal with past events, not future events. In addition, they only deal with actions that affect the database. In contrast, in planning, we deal with actions that are intended to be executed in the future, these actions have an effect on the real world, and these effects are represented in the database by making updates to the database at appropriate future time instances. This involves notions like coherence and consistency that are not relevant for transaction time (notions of consistency associated with database locking are very different). Valid time usually associates with an ordinary relational tuple, either a single time stamp, or a time interval. These denote the time when an event is true (or a time interval throughout which the event is true). Even though the start and end times of actions can be stored in a temporal database, temporal databases do not reason about the effects of these actions and allow queries that require reasoning about such effects.

There are also a few pieces of work [5,3] involving non-deterministic time, in which one can make statements of the form "An event is valid at some time point in a given interval" (as compared to being true throughout the interval). Consistency here can be important [10,7]. Users might be interested in temporal queries such as "Find all events starting after some time t or after completion of some other event e." Processing such queries requires checking consistency of temporal constraints. In such temporal constraint databases and query languages, the temporal constraints used can be much more expressive than those used in our model. However, the purposes are very different. These works dis-

oarieli@mta.ac.il

cuss the occurrence of events at time points in the future, but not about the fact that these events could be actions that have an impact on the world. As a consequence, they do not model the fact that their events can trigger updates to the database. Hence, there is no need in their frameworks for concepts like consistency, coherence, and closure introduced here, and our definitions of the algebraic operations are correspondingly different.

6 Conclusions

Many agencies and corporations store complex plans—ranging from production plans to transportation schedules to financial plans— composed of hundreds of "interlinked" plan elements. Such applications require not only that plans be created automatically, but also that they be stored in an appropriate data model, and that they be monitored and tracked during as they (i.e. the plans) are executed. To date, most work on plans has focused on the *creation* of plans.

In this paper, we propose a *data model* for storing plans so that plans may be monitored and tracked. We propose the concept of a *plan database* and provide algebraic operations to query such databases. These algebraic operations extend the classical relational operations of selection, projection, join, etc. In addition, we provide algorithms to update sets of plans as new plans need to be added to the database, and as old plans are executed (and either adhere or do not adhere to their intended schedules).

Much future work remains to be done on plan databases, and this paper merely represents a first step. Topics for future study include scalable disk-based index structures to query plan databases, cost models for plan algebra operations, equivalences of queries in plan databases, and optimizing queries to plan databases.

Acknowledgments. This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, DAAL0197K0135 and DAAD190320026, Naval Research Laboratory N00173021G005, the CTA on Advanced Decision Architectures, ARO contracts DAAD190010484 and DAAD190310202, DARPA/RL contract number F306029910552, NSF grants IIS0222914 and IIS0329851 and the University of Maryland General Research Board. Opinions expressed in this paper are those of authors and do not necessarily reflect opinion of the funders.

References

1. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, New York, 1995.
2. J. Chomicki. Temporal query languages: a survey. In *Temporal Logic: ICTL'94*, volume 827, pages 506–534. Springer-Verlag, 1994.
3. C. E. Dyreson and R. T. Snodgrass. Supporting valid-time indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, 1998.

oarieli@mta.ac.il

4. M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains, 2002. http://www.dur.ac.uk/d.p.long/pddl2.ps.gz.
5. S. Gadia, S. Nair, and Y. Peon. *Incomplete Information in Relational Temporal Databases*. Vancouver, 1992.
6. K. J. Hammond. Case-Based Planning: Viewing planning as a memory task (Academic Press, San Diego, CA, 1989).
7. M. Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141–173, 1994.
8. D.-T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12):745–758, 1997.
9. R. Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems (TODS)*, 12(2):247–298, 1987.
10. P. T. V. Brusoni, L. Console and B. Pernici. Qualitative and quantitative temporal constraints and relational databases:theory, architecture and applications. *IEEE TKDE*, 11(6):948–968, 1999.

oarieli@mta.ac.il

Author Index

oarieli@mta.ac.il