Dagstuhl Report on

Logic Databases and the Meaning of Change

Compiled by
Andrei Voronkov
Uppsala University
Computing Science Department
Box 311
S 75105 Uppsala, Sweden
<http://www.csd.uu.se/~voronkov>
<voronkov@csd.uu.se>

1 A Note from the Organizers

Hendrik Decker
<Hendrik.Decker@mchp.siemens.de>

This report gives account of the seminar on *Logic Databases and The Meaning of Change*, held at the marvelous conference and research center of Dagstuhl, in the last week of September 1996. The seminar attracted a total of 36 invited researchers in the field, from 14 countries. Each of the participants contributed to the seminar by giving talks, demonstrations and/or participating in lively discussions which followed upon every presentation. Particular highlights included talks on the Situation Calculus (Monday), on Transaction Logic (Tuesday), on Active Databases (Wednesday), as well as on Abductive, Inductive and other, more involved inference techniques. Not surprisingly, the most frequently addressed topics are named by the keywords Updates, Transactions, and Integrity.

Often, discussions after presentations did not cease during breaks for coffee and lunch, and always continued well into the evenings. Exchanges of opinions and points of view were favorably stimulated by the well-appreciated personal atmosphere of the by-now famous Dagstuhl wine cellar and other locations in the Schloss. Apart from many productive sessions in the wine cellar, some particularly vivid debates and interchanges of views were witnessed during several open forum discussions, scheduled on the last two days

of the seminar week. Topics were *The Frame Problem*, *View Updating* and, finally, *Future Directions and Expectations* in the field. Originally, the plan was to have panels with statements of contending convictions by selected panelists. However, it turned out that, fortunately, no separation between opinion leaders and followers was needed. Rather, several contrasting or sympathetic beliefs were spoken out freely by a large number of participants. No general consensus was achieved (of course, not), but many felt enriched after having been exposed to a rich diversity of very interesting, engagingly articulated points of view.

Besides the very well-received session with demonstrations of implemented knowledge-based systems, agreeable divertissement was provided for by the numerous free time facilities at the Dagstuhl center, as well as by an enjoyable excursion to the Saarschleife (a near-by natural sight-seeing area), which was followed by a visit to the Villeroy & Boch ceramics manufacturing plant.

Many thanks are due to the friendly staff at the Dagstuhl office in Saarbrücken as well as at the Schloss, for everything they have done to enable a smooth organization of the seminar week, to establish a perfect ambiance at their idyllic location, and to prepare some excellent food.

An on-line version of this report is available at the URL
`<http://www.uni-koblenz.de/ag-ki/dag9639/>`.

An on-line version of extended abstracts is also available at the URL
`<ftp://ftp.csd.uu.se/pub/papers/reports/0129.ps.gz>`.

2 Letter of Invitation

Hendrik Decker
`<Hendrik.Decker@mchp.siemens.de>`

Logic databases have been an exciting, dynamic research field since the '70s. Important results were achieved in the areas of query processing, constraints, semantics, and several successful prototypes have been developed. However, most of this work was concerned with *static* aspects of data representation and querying. So, when confronted with practical needs, the existing systems had to find ingenious (albeit somewhat parochial) remedies to tackle the *dynamics* of logic databases, i.e., updates to database states.

Although transactions and updates permeate any database system worth its salt, the consensus on how to treat updates in logic data and knowledge bases is slow to come. In fact, transactions are notoriously awkward and hard-to-understand parts of logic-based programming systems, which is a serious obstacle to gaining a foothold in traditional and emerging areas of

information processing. It is also a blemish on a technology that purports to provide a solid methodology for building complex, robust and intelligent systems.

Fortunately, a number of solid approaches are beginning to emerge, and time is ripe to size them up. The seminar on *Logic Databases and the Meaning of Change* is intended to bring together key researchers in the field. The organizers confidently expect that the idyllic atmosphere of Schloss Dagstuhl will provide for productive, stimulating discussions on the foundation of state changes in data and knowledge bases.

A sound but incomplete list of issues to be addressed includes:

- abduction;
- active logic databases;
- consistency and integrity;
- dynamics of logic and object-oriented systems;
- evolution and versioning;
- hypothetical query answering;
- intensional query answering;
- schema transformation;
- logical transactions and updates.

3 Abstracts of the talks

Following is the abstracts of the talks given at the seminar. Wherever possible the on-line home page and email addresses of the speakers are included for reaching them easily. Many speakers have provided an on-line reference to a paper related to their talks. These abstracts and other information about this Dagstuhl seminar are available on the web at the URL `<http://www.uni-koblenz.de/ag-ki/dag9639/>`.

Chandrabose Aravindan
<http://www.uni-koblenz.de/~arvind/>
<arvind@informatik.uni-koblenz.de>

Universität Koblenz-Landau
Fachbereich Informatik
Rheinau 1, D-56075 Koblenz, Germany

On the rationality of knowledge base dynamics

In this talk, we first motivated the need to bring together two independent tracks on the notion of change: belief dynamics and algorithms for concrete cases of change. We also argued that to apply the rationality results of belief dynamics theory to various practical problems, it should be generalized in two respects: first of all, it should allow certain part of the belief to be declared immutable; and secondly, the belief state need not be deductively closed. Such a generalization of belief dynamics, referred to as knowledge base dynamics, was then presented along with the concept of generalized contraction. We then showed that knowledge base dynamics has interesting connections with abduction, thus enabling us to use abductive procedures to realize contraction. Finally, we demonstrated how knowledge base dynamics can provide an axiomatic characterization for deleting view atoms from databases.

This report is available on-line at
http://www.uni-koblenz.de/~arvind/papers/jancl-rev.ps.gz

Patrizia Asirelli
<http://rep1.iei.pi.cnr.it/people/asirelli>
<asirelli@iei.pi.cnr.it>

Istituto di Elaborazione della Informazione - C.N.R.
Via S. Maria, 46
56126 Pisa, Italy

Integrity Constraints versus View Constraints

The problem of Integrity Constraints checking in the area of logic databases is considered. On one hand, constraints can be considered as 'obligations' (Integrity Constraints). Another possible interpretation of the same formulas is to view them as defining subsets of a given database (View Constraints).

The particular way of considering the role of constraints, as is apparent from the various methods in the literature, is shown to be often an issue of pragmatics, depending on the application at hand. Two examples of application of the two concepts of Integrity Constraints and View Constraints are given. The first one shows that this kind of constraints can be used to check

the well-formedness of a net in the Extended Transition Networks formalism. The second example is an application for the definition of a constraint operator among logic programs.

Two implementations of the notion of View Constraints are sketched. One refers to an earlier paper (JLP1985 - M. Martelli, M. De Santis and myself), where a new database state is computed on the basis of the old one and the Integrity Constraints. The second implementation takes a more dynamic view: It defines a new refutation procedure (SRP-IC) which is based on SLD. Every answer to a query which is computed by SRP-IC is successful if and only if it is entailed by the database and satisfies the constraints.

Jorge Bocca
<Jorge.Bocca@ids.de>

IDS Integral Decisions Systems SE GmbH
Belfortstr 6+8
D-81667 München, Germany

A Deductive Executable Repository for Data Mining

This paper presents the architecture for the slave abstract machine for the DAFS system. It addresses the requirements for functionality and performance for DAFS, as identified in other reports in the DAFS project. For this, it presents the technological background leading to a kernel architecture for a single processor slave unit. It goes further to discuss the options available to provide solutions to the open issues left by the kernel architecture. It finally, indicates the approach to be adopted in addressing these issues.

Laurence Cholvy
<http://www.cert.fr/francais/deri/cholvy/page.html>
<cholvy@cert.fr>

ONERA-CERT
2 avenue Ed Belin
31055 Toulouse, FRANCE

Adding information by taking topics into account

In this work, we contribute to define what becomes a database after the addition of a piece of information.

We show that the result of the addition trivially depends on the current state of the database, on the new information, but also depends on the topics

of information. More precisely, it depends on the relative reliability, topics by topics, of the agents who store data in the database.

We consider that a database is a set of beliefs some agents have about the real world. Adding a piece of information then means adding some beliefs of some agents.

We show that, in the classical database approach, the semantics of the addition is based on a postulate which states that the current database state is always preferred to the new information. Indeed, when the new information contradicts the current state, its addition is rejected. This comes to say that the agents who stored the information in the database are more reliable than the agent who wants to store the new information.

At the opposite, in the belief revision approach, the semantics of the addition is based on a postulate which states that the new information is always preferred to the current state. Indeed, when the new information contradicts the current state, the current state must be revised (i.e., some information must be deleted in it) in order to add the new information. This comes to say that the agent who provides the new information is more reliable than the one who have provided the stored information.

The approach we suggest is an intermediate one : we postulate that, the preference between the current state and the new information depends on the topics of information : for a given topic, the current state may be preferred to the new information, but for some other topic, it can be the contrary. In other terms, agents are more or less reliable, according to the topic.

We present a belief logic, based on KD, which allows us to reason with information provided by agents who are more or less reliable, according to the topic.

Henning Christiansen
<www.dat.ruc.dk/~henning>
<henning@dat.ruc.dk>

Roskilde University
Computer Science Dept.
P.O. Box 260
DK-4000 Roskilde

Metainterpreters for modelling change in logic databases

A traditional interpreter for logic programs works as a query answering mechanism for logic databases and taking the step to metainterpretation makes it possible to model "higher-order" actions such as update and change.

A metainterpreter for a logical language defines a proof predicate, which

(depending on language and application) approximates or is identical to a truth predicate.

Interesting properties can be modelled by declarative statements involving truth (or provability) and the use of a suitable metainterpreter provides automatically an implementation. This may require the metainterpreter being able to work "backwards" in the sense that it can generate programs or program fragments in order to make specified goals provable. We show applications of a constraint-based metainterpreter with this capability for modelling update via views and induction of intensional rules.

Furthermore, metainterpretation can be used for modelling nonstandard derivation rules, possibly controlled in various ways and, thus, with the possibility of going beyond a strictly logical semantics. This is relevant in flexible or cooperative query answering systems, where hypothetical updates and automatic change of query (or subgoal) can be used as ways to provide answers better suited for the user's needs — which may or may not be expressed explicitly in the query.

The use of metainterpreters as a complement to traditional mathematical nomenclature provides executable and easy-to-modify specification, which may serve directly as implementations or, perhaps more realistically, as prototypes for efficient implementations.

Hendrik Decker
<Hendrik.Decker@mchp.siemens.de>

Siemens ZT AN 4
D-81730 München, Germany

View Updating by Abductive Logic Programming

We present an extension of the SLD resolution proof procedure by two key features of Abductive Logic Programming (ALP). Firstly, subsidiary computations are employed for abducing consistent hypotheses. Secondly, we make active use of denial integrity constraints as input clauses of forward reasoning steps. The resulting procedure, called SLDAI, improves existing ALP procedures in three ways:

1. *Reasoning forward from abducible hypotheses through literals of opposed polarity is facilitated by a very simple but effective new inference rule, which derives conclusions of the form*

$$\neg A\theta \leftarrow \neg A\theta$$

from premises of the form

$$B \leftarrow \qquad \textit{and} \qquad A \leftarrow B' \& C$$

where B and B' are literals of opposed polarity such that the atoms of B and B' unify by some mgu θ.

2. *Hypothesis formation in SLDAI is eager (i.e., several new hypotheses may be assumed at a time), as opposed to the less effective lazy hypothesis formation in known ALP procedures (which may assume at most one new hypothesis at each step).*

3. *Unnecessary hypothesis formation, which often fatally hampers known ALP procedures, is avoided in SLDAI.*

A first version of SLDAI has been reported on by this author in Proc.s JICSLP'96, pages 157 - 169, MIT Press, 1996 (Editor: M. Maher).

Stefan Decker
<http://www.aifb.uni-karlsruhe.de/Staff/sde.html>
<decker@aifb.uni-karlsruhe.de>

Institut AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany

Structuring SLDNF-Proofs. Principles and Applications

Change in deductive databases usually means deletion or addition of new facts and rules. We aim at understanding the effects of an update by structuring and analysing a SLDNF-proof, so the exact consequences of an update relative to a proof can be determined. The approach is interesting for the field of integrity checking in deductive databases: better conditions for an update violating the proof of an integrity constraint can be defined.

For structuring an SLDNF-proof we start with an and-or-tree. ¿From a logic program and a query an and-or-tree can be built up in the usual way: starting from the goal clause (an and-node) the successors of an and-node (the or-nodes) are the atoms, that are used in the literals of the clause. If a literal is negative the edge from the clause to the atom is marked with a negation sign. In general an and-or tree has an infinite size (due to recursive clauses). If a proof system (like PROLOG) finds a proof this proof has always a finite size. Adding this substitutions to the tree, the parts of the tree, that are not needed for the proof, can be identified.

We have two main possibilities to exploit a prooftree in the problem of incremental integrity checking: Often the storing of all substitutions might be too costly In this case it is possible to store only the relevant rules of the deductive database without further considering the substitutions. Because we know all relevant rules and literals of a proof we can easily determine, when an update influences the validity of the proof. If it is possible to store the substitutions algorithms exists, that realize an effcient JTMS for normal logic programs.

This structuring method can be used as a framework for integrity checking in deductive databases. Most known approaches in integrity checking can be related to this general method. However, a number of questions remain unanswered, eg.: How can the computation rules for which a construction of a prooftree is possible be characterized? How is a prooftree related to optimized evaluation strategies for deductive databases (magic sets etc.)? Answering this question would give another relation of bottom-up and top-down evaluation strategies.

Peter A. Flach
<http://machtig.kub.nl:2080/infolab/Peter/Peter.html>
<Peter.Flach@kub.nl>

Katholieke Universteit Brabant
INFOLAB
P.O. Box 90 153
5000 LE Tilburg, The Netherlands

Induction in the context of Logic Databases

In this paper I argue that Inductive Logic Programming (ILP), a recently established subfield of Machine Learning that induces first-order clausal theories from examples, combines very well with the area of Logic Databases. In the context of Logic Databases, induction can be defined as inference of intensional knowledge from extensional knowledge. Classification-oriented ILP approaches correspond to induction of view definitions (IDB rules), while description-oriented ILP approaches correspond to induction of integrity constraints. Possible applications of ILP methods in Logic Databases thus include induction of IDB rules and learning of integrity constraints. Further possible applications include intensional query answering, database restructuring, and induction of database dynamics such as update rules. The paper gives a general overview of ILP with particular emphasis on possible application of

ILP methods in Logic Databases.

Burkhard Freitag
<http://www.uni-passau.de/~freitag/>
<freitag@fmi.uni-passau.de>

Universität Passau
Fakultät für Mathematik und Informatik
94030 Passau
Germany

Using Hypothetical States to Implement Bulk Updates and Sequential Updates

Transaction processing requires that the effects of a non-terminated transaction can be undone in case of a failure. In databases, this is achieved by deferring the updates caused by a transaction until 'Commit', or, in other words, by introducing hypothetical database states that can be accessed exclusively by the database operations issued by this transaction and materializing the changed database after the end of the transaction. Although in databases all this is normally done at the physical level, the idea of deferring updates and maintaining hypothetical states carries over to deductive database programs containing updates.

The semantics of the rule-based database programming language presented in the companion paper by the same authors is based on the concept of sets of update requests which potentially cause the transition from some initial database state to some new state. We present a transformation of update programs into conventional deductive database programs which maintain update request sets in an appropriate way. Essentially, every access to the database state is redirected to a hypothetical current state that is represented by the current set of update requests. Furthermore, every transformed update rule generates the corresponding set of new update requests that, if applied to the current state, would result in the new database state. Finally, for every extensional predicate a pair of frame axioms is generated that formalize the hypothetical state of the extensional relations. The current and new update request sets can be viewed as an encoding of (local) transaction logs. However, an implementation has to observe that, in analogy to conventional transaction processing, a global transaction log is required for the synchronization of concurrent transactions.

We also show that representing the current and next database states by explicit state terms containing as arguments the appropriate sets of update requests leads to a situation-calculus-like formulation of the update rules

and the frame axioms. Although for closed world databases, e.g. relational databases, a clause-based logic with negation-as-failure is sufficient, this reformulation indicates a possible way to handle also open world databases.

This is a joint work with Carl-Alexander Wichert

Burkhard Freitag
<http://www.uni-passau.de/~freitag/>
<freitag@fmi.uni-passau.de>

Universität Passau
Fakultät für Mathematik und Informatik
94030 Passau
Germany

Logical Specification of Bulk Updates and Sequential Updates
See the entry for Carl-Alexander Wichert

Fosca Giannotti
<f.giannotti@cnuce.cnr.it>

CNUCE Institute of CNR
Via S. Maria 36
56125 Pisa, Italy

A Logic Abstract Machine for Active Deductive Object-Oriented Databases

This paper tries to identify an abstract machine capable of supporting the mechanisms of an active/deductive object-oriented data model. We are interested in finding an appropriate abstraction level for a twofold purpose. On one side, such a machine should provide an abstract implementation level, where the object model is declaratively reconstructed, and its integration with actions and deductions clarified. On the other side, the machine should provide the basis of a realistic implementation, at least at a prototype level.

In this paper, we consider a natural extension of Datalog with a single unary function and a non-deterministic construct, which corresponds to a fragment of the logical data language LDL++, as a candidate intermediate language for constructing the mentioned abstract machine. Next, we consider a small language embodying the essential aspects of an active/deductive object data model, and provide a compilation *of this language into the mentioned fragment of LDL++. Such fragment is high-level enough to allow a formal specification of the crucial aspects, such the design of the object store, the*

inheritance mechanisms, the deductive query processing, the semantics of active rules, and the integration of these various competing paradigms in a coherent framework.

Moreover, the proposed compilation is low-level enough to form the basis of a realistic (prototype) implementation, as LDL++ can be efficiently executed by means of a fixpoint procedure with real side-effects to support updates.

The static component of the considered object model, i.e., its data definition and query language, can be seen as a fragment of F-logic*, and provides mechanisms for multiple inheritance, multiple roles, virtual objects, views, and methods (derived attributes). The dynamic component of the considered object model, i.e. the basic update and object-migration operations, is inspired by the analogous features of the Fibonacci language, but comprises also a simple form of active rules, or* condition-action *rules.*

The compilation uses non determinism to realize object identifiers, and the non-deterministic semantics of active rules, and a form of stratification to realize updates, object-migration and, in general, actions against the database.

The code obtained as a result of the compilation can be understood in declarative terms, thus providing a formal interpretation of the source object-oriented program. But more importantly, such a code can be executed by the ordinary deductive, fixpoint-based computation integrated with efficient support for updates by means of side effects.

This is a joint work with M. Carboni, G. Manco and D. Pedreschi

Laura Giordano
<http://www.di.unito.it/pub/www/LP/laura.html>
<laura@di.unito.it>

Dipartimento di Informatica
Università di Torino
C.so Svizzera 185
10149 Torino, ITALY

From Hypothetical Reasoning to Reasoning about Actions in Modal Logic Programming

In [2] we have proposed a logic programming language CondLP$^+$*, that is an extension of N_Prolog [1] which provides capabilities for hypothetical and counterfactual reasoning. In* CondLP$^+$*, a database containing constraints can be hypothetically updated by adding new facts. A revision mechanism allows to deal with the inconsistencies that may arise by removing the information responsible for them.*

In this language, as in N_Prolog, hypothetical implications $D \Rightarrow G$ (with G a goal and D a set of hypotheses) may occur both in goals and in clause bodies. For an implication $D \Rightarrow G$ to succeed from a program P, G must succeed from the new program obtained by hypothetically updating P with D. As a difference with N_Prolog, in CondLP$^+$ *a program may contain integrity constraints, and, hence, inconsistencies may arise, when updates are performed.*

In CondLP$^+$ *hypothetical updates are sets of atoms. Moreover, a program consists of a protected part Δ, containing a set of clauses and integrity constraints, and a removable part, consisting of a set of atomic formulas partitioned in different priority classes. When an inconsistency arises, a* revision *of the program is needed. Revision does not affect the protected part, which is permanent. Instead, it modifies its removable part, by overruling the atoms which are responsible for the inconsistency and have the least preference.*

For this language a logical characterization has been defined by making use of a (preaty standard) modal logic, in which hypothetical updates are represented by modalities. In fact, modal logic allows very naturally to represent state changes as transition, through the accessibility relation of Kripke structures. Moreover, to capture the non-monotonic behaviour of the language. an abductive construction is introduced. More specifically, persistency axioms are introduced to allow persistence of assumptions, if they are not inconsistent with other assumptions with higher priority.

By generalizing the proposed approach, modalities can be used to represent actions rather than updates (on the line of other proposals for reasoning about actions which are based on Dynamic Logic [3]). This leads to the definition of a modal language for reasoning about actions, in which abduction is used to deal with the frame problem.

References

[1] D. M. Gabbay and N. Reyle. NProlog: An extension of Prolog with hypothetical implications.I. Journal of Logic Programming*, (4):319–355, 1984.*

[2] Dov Gabbay, Laura Giordano, Alberto Martelli, and Nicola Olivetti. Hypothetical updates, priority and inconsistency in a logic programming language. In Proc. LPNMR-95*, LNAI 928, pages 203–216, 1995.*

[3] D. Harel. First order dynamic logic in Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984.*

Anna Gomolińska
<http://www.math.uw.bialystok.pl/~anna.gom>
<anna.gom@math.uw.bialystok.pl>

Bialystok Branch of Warsaw University
Institute of Mathematics
Akademicka 2
PL - 15-267 Bialystok
Poland

Multi-sorted Preferences and Belief Change

In the presented framework the issue of changing a belief state is addressed from the perspective of multi-sorted preferences upon given information. Most of the research in the area of modelling a belief change is influenced by the joint work of Alchourrón, Gärdenfors, and Makinson who developed the theory of belief revision. *Belief states are represented by* belief sets*, i.e., sets of formulas closed under logical consequence. Changing a belief state is described in terms of certain transformations, viz.,* expansion*,* revision*, and* contraction *on a given belief set when facing new information.*

It turns out that these transformations can be defined by means of a preordering on formulas called an epistemic entrenchment *relation (in short, an EE-relation). An EE-relation represents preferences on formulas in case some of them have to be removed from a belief set.*

When revising and/or contracting a belief set by a formula, a large number of candidates for the new belief set can be produced even if an EE-relation was previously employed to select the best among them. From the well-known strategies to choose the best one, viz., Maxi-choice, Full-meet, and Partial-meet, only the last one is considered as satisfactory provided that an appropriate selector is given.

In our approach such a selector is proposed by means of a multi-sorted preference relation. In case of Maxi-choice and Full-meet strategies the number of suitable candidates can be reduced considerably. Instead of an EE-relation only, several preference pre-orderings are considered. The sorts of preference of a particular interest are credibility, utility, importance, and being logically derivable. The preference relations are combined to a multi-sorted preference pre-ordering. The definition of a belief set is modified accordingly to the multi-sorted case by means of certain consequence operators associated with the sorts of preference. A belief change process based on multi-sorted preferences takes in particular forms corresponding to extension,

revision, and Hansson's consolidation.

References

[1] Alchourrón, C. E., Gärdenfors, P., and Makinson, D.: On the Logic of Theory Change: Partial Meet Contraction and Revision Functions, Journal of Symbolic Logic **50***, 1985, 510 – 530.*

[2] Hansson, S. O.: Belief Base Dynamics. Doctoral dissertation at Uppsala University, 1991.

Knut Hinkelmann
<http://www.dfki.uni-kl.de/~hinkelma>
<hinkelma@dfki.uni-kl.de>

DFKI GmbH – German Research Center for Artificial Intelligence
Postfach 2080
67608 Kaiserslautern, Germany

Refinement Operators for Structured Logic Databases

When changing a knowledge base a general distinction is made between revision *and* update *[5,4]: an update helps a knowledge base to keep up-to-date with an evolving world while a revision is a belief change where information is added or removed about a static situation of which not everything is known.*

In a structured logic database the knowledge is composed of the extensional database (EDB – a set of ground facts on base relations), the intensional database (IDB – a set of rules), and the ontology, usually containing a sort theory and an integrity theory.

It is often assumed that "the rules are generally regarded to be complete and non-defeasible and thus should not be changed, whereas the EDB is regarded as containing incomplete and defeasible knowledge about the world" [2]. Consequently, it is argued that update *operations should only affect the base relations in the EDB (cf. [1]). However, there are applications where this assumption does not necessarily hold: it is easy to find examples where both* revision *and* update *operations can affect all parts of the logic database: the EDB, the IDB, or the ontology.*

In the paper we will consider two approaches tackling the problem of curing update-triggered inconsistencies in structured logic databases, both sharing these interesting characteristics: (1) an inconsistency can be cured by changing any part of the logic database: EDB, IDB, or ontology; (2) the refinement operators exploit information provided by the structuredness of the logic database; (3) inference mechanisms used to answer plain queries do not have to be extended but are simply reused.

1. *We present a transformational approach for integrity checking that suggests hypotheses for refinement operations. This technique can be applied if an EDB-update causes an inconsistency. In addition to the removal or addition of facts or literals the approach allows for new revision operators like specialization and generalization of sorts and builtin operators (like comparison operators). The transformation proceeds in three steps: (1) unfolding the integrity theory, (2) generating trigger rules for each possible update, (3) extending the trigger rules by revision hypotheses.*

2. *If a logic database contains rules that were detected by data mining, they cannot be assumed to be complete and non-defeasible, since they depend on defeasible training data. To keep these rules up-to-date with changing data sets we show how techniques from Inductive Logic Programming (ILP [3]) can be applied not only for learning new rules but also for specializing already existing rules. We present sort restriction as a refinement operator, sketch several techniques for avoiding irrelevant refinements, and discuss the reuse of intermediate results for more efficient satisfiability and consistency tests.*

References

[1] Hendrik Decker. An extension of SLD by abduction and integrity maintenance for view updating in deductive databases. In International Joint Conference and Symposium on Logic Programming, Proceedings*, 1996.*

[2] A. C. Kakas and P. Mancarella. Database updates through abduction. In Proceedings of the 16th International Conference on Very Large Databases (VLDB)*, 1990.*

[3] N. Lavrač and S. Džeroski. Inductive Logical Programming: Techniques and Applications*. Ellis Horwood Limited, 1994.*

[4] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In James Allen, Richard Fikes, and Erik Sandewall, editors, Principles of Knowledge Representation and Reasoning – Proceedings of the Second International Conference (KR91)*, pages 387–394, San Mateo, CA, 1991. Morgan Kaufmann Publishers.*

[5] Léa Sombé. A glance at revision and updating in knowledge bases. International Journal of Intelligent Systems*, 9:1–27, 1994.*

This is a joint work with Andreas Abecker and Michael Sintek.

Georg Lausen
<http://www.informatik.uni-freiburg.de/~lausen>
<lausen@informatik.uni-freiburg.de>

Institut für Informatik
Universität Freiburg
Am Flughafen 17
79110 Freiburg, Germany

Declarative Semantics of Referential Actions

The concept of referential integrity plays an important role in existing relational database systems. Referential actions *(*rac's*) are specialized triggers (active rules) which are used to automatically maintain referential integrity constraints by local "repairs" after a user-requested update. Since rac's allow to execute or to block deletions, their semantics is inherently nonmonotonic. While the local effects of rac's are quite intuitive and easy to grasp, the global effect of several rac's acting together may give rise to ambiguities if no global semantics for rac's is specified. The SQL2 and SQL3 standards include rac's, but describe their meaning in a lengthy, operational, and semi-formal way making it very difficult to understand or predict the global behavior of a set of rules.*

In contrast, we propose to specify a set A of rac's of the form `ON DELETE {CASCADE,RESTRICT,NO ACTION}` *by mapping them to a normal logic program* P_A*. We show that the nonmonotonic semantics developed for logic programs lead to a concise and elegant, yet strictly formal specification of rac's. In particular, ambiguities due to "conflicting" rac's become apparent in the* well-founded model *of* P_A*, since they correspond precisely to undefined atoms. The* stable models *of* P_A *represent alternative policies of resolving these ambiguities. There always exist two distinguished stable models which can be efficiently computed using* Statelog*, a state-oriented extension of Datalog. Additional insight into the behavior of rac's is given by a* game-theoretic interpretation *of the well-founded model of* P_A*.*

This is a joint work with Bertram Ludäscher and Wolfgang May

Michael Leuschel
<http://www.cs.kuleuven.ac.be/~michael>
<michael@cs.kuleuven.ac.be>

Departement Computerwetenschappen
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Specialised Integrity Checking by Combining Conjunctive Partial Deduction and Abstract Interpretation

A meta-program is a program which takes another program, the object *program, as one of its inputs. An important issue in meta-programming is the representation of the object program. One approach is the* ground representation*, which encodes variables at the object level as ground terms. A lot of meta-programming tasks can only be written declaratively using the ground representation. This is e.g. the case for the application in [1], where a simplification procedure for integrity constraints in recursive deductive databases is written as a meta-program. The goal is to obtain a pre-compilation of the integrity checking via partial deduction of the meta-interpreter. However, contrary to what one might expect, partial deduction is then unable to perform interesting specialisation and no pre-compilation can be obtained. Indeed, the ground representation entails the use of an explicit unification algorithm at the object level and then current abstract interpretation methods, as well as current partial deduction methods alone, fail to derive crucial information.*

We will present an elegant and powerful solution to this problem by combining two existing analysis schemes, each underlying a specific specialisation technique: the one of conjunctive partial deduction *and the one of* more specific programs*. By this approach, also detailed in [2], we will be able to obtain the required specialisation, as well as an analysis which surpasses the precision of current abstract interpretation techniques.*

References

[1] M. Leuschel and B. Martens. Partial deduction of the ground representation and its application to integrity checking. In J. Lloyd, editor, Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 495–509, Portland, USA, December 1995. MIT Press. Extended version as Technical Report CW 210, K.U. Leuven. Accessible via* `http://www.cs.kuleuven.ac.be/~lpai`.

[2] M. Leuschel and D. Schreye. Logic program specialisation: How to be more specific. In H. Kuchen and S. Swierstra, editors, Proceedings of the International Symposium on Programming Languages, Implementations,

Logics and Programs (PLILP'96), *LNCS 1140, pages 137–151, Aachen, Germany, September 1996. Extended version as Technical Report CW 232, K.U. Leuven. Accessible via* `http://www.cs.kuleuven.ac.be/~lpai`.

This is a joint work with Danny De Schreye.

Danilo Montesi
`<http://www.sys.uea.ac.uk>`
`<dm@sys.uea.ac.uk>`

School of Information Systems
University of East Anglia
Norwich NR4 7TJ, UK

Integrity Constraints Evolution

Integrity constraints are usually assumed to be permanent properties that must be satisfied by any database state. However, there are many situations requiring also temporary constraints, that is, constraints that must hold only for a single database state. We propose a schema to define both permanent and temporary constraints, that supports efficient constraint checking and semantic query optimization. The proposed schema associates integrity constraints to rules and queries, and it uses methods, that were originally defined for permanent constraints only, to perform constraint checking and semantic query optimization. The proposed schema is also promising to link basic update execution with constraint checking. Indeed, the schema allows to define local integrity constraints with respect to a rule. Local constraints are defined within a rule and thus if updates are defined in the same rule, then they can trigger the constraint checking.

This is a joint work with Chiara Renso and Franco Turini, Dipartimento di Informatica, Universitá di Pisa, Italy

Hans Nilsson
`<http://www.ericsson.se/cslab/~hans>`
`<hans@erix.ericsson.se>`

Ericsson Telecom AB
Computer Science Laboratory
S-125 26 Stockholm, Sweden

Mnesia — An Industrial Deductive DBMS with Transactions and Distribution

Mnesia *(previously Amnesia) is a multiuser Distributed DBMS specially made for industrial telecommunications applications written in the symbolic functional real-time language Erlang [1]. All of Mnesia is written in Erlang except some low level C support for table hashing.*

The applications we now see implemented in the Erlang language needs: (1) fast soft real-time key-value lookup, (2) non-realtime queries, (3) distribution, (4) fault-tolerance and (5) dynamic re-configuration.

Mnesia provides this with primary memory storage, distributed transactions, fast lookups which may bypass the transactions, optional replication of data on separate nodes or disks, location transparency and a deductive query language integrated in Erlang with list comprehensions. *The table properties (e.g location, replication) may be altered in runtime.*

Queries are compiled, optimized and evaluated as operators [3]. Recursion is handled by SLG-resolution.[2].

References

[1] Joe Armstrong, Robert Virding, Claes Wikström and Mike Williams. Concurrent Programming in ERLANG. *Prentice Hall, second edition, 1995.*

[2] A. W. Chen and D. S. Warren. Query evaluation under the well-founded semantics. In Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, page 168, Washington, DC, May 1993.*

[3] Goetz Graefe. Query evaluation techniques for large databases. ACM Computing Surveys*, 25(2):73–170, June 1993.*

This is a joint work with Claes Wikström

Werner Nutt
`<http://www.dfki.uni-sb.de/~nutt>`
`<nutt@dfki.uni-sb.de>`

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3
D-66123 Saarbrücken
Germany

Taxonomies and Classification Rules for Semi-structured Data

Traditional database techniques presuppose that information is representable in a uniform and rigid format. However, there are applications where information is stored that has some structure but is not formatted as rigidly and regularly as items in relational or object-oriented data bases. Examples for such semi-structured data *are:*

- *files, like bibliographies on the web or biological databases*
- *html-documents, which contain structuring primitives like header, host, enumerations etc.*
- *data models for the integration of heterogeneous information sources.*

In these cases, data evolve too quickly or without sufficient coordination, so that any attempt to describe a given situation in terms of a standard data model will necessarily lag behind. Semi-structured data have recently become a research topic, and data models and query languages for them have been proposed [5,1].

Models for semi-structured data are highly flexible so that they can capture heterogeneous and changeable information. However, they lack facilities to convey to a user the information content of the modeled data. We propose to add as an additional layer on top of a such a data model a taxonomy of classes. The classes describe the kinds of objects occurring in a domain, and objects can be accessed through a class. The classes are populated by classification rules *that specify sufficient conditions for an object to be an instance of a class. The classes are not organized in a static taxonomy, but their containment relations can be derived by automatically reasoning about the corresponding rules.*

The rule languages we consider are not matched by any of the languages studied in the database area. However, as we show, there is a close connection between data models and query languages for semi-structured data on the one hand and description logics in Artificial Intelligence on the other hand. Description logics are a family of languages that are used in Artificial Intelligence to represent knowledge about objects and classes of a domain. More precisely, we show that classification rules are related through a simple duality principle to so-called inclusion constraints in description logics [2,3,4]. The duality allows us to reinterpret results established in one area in terms of the other. In particular, we exploit it to determine the complexity of checking the containment of classes for an entire family of rule languages.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. Technical report, The Stanford University Database Group, 1996.

[2] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. Refining the structure of terminological systems: Terminology = Schema + Views. In Proc. 12th National Conference on Artificial Intelligence*, pages 199–204, Seattle, (Washington, USA), August 1994. Morgan Kaufmann Publishers.*

[3] M. Buchheit, F.M. Donini, W. Nutt, and A. Schaerf. A refined architecture for terminological systems. Research Report RR-95-09, DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, June 1995.

[4] Diego Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In W. Wahlster, editor, Proc. 12th European Conference on Artificial Intelligence*, pages 303–307, Budapest (Hungary), August 1996. John Wiley and Sons.*

[5] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In Tok Wang Ling, Alberto O. Mendelzon, and Laurent Vieille, editors, Proc. 4th International Conference on Deductive and Object-Oriented Databases*, volume 1013 of* Lecture Notes in Computer Science*, pages 319–344, Singapore, December 1995. Springer-Verlag.*

This is a joint work with David Michaeli and Yehoshua Sagiv

Raymond Reiter
<http://www.cs.toronto.edu/~cogrobo/>
<reiter@cs.toronto.edu>

University of Toronto
Dept. of Computer Science
10 King's College of Road
Toronto Ontario M5S 1A4, Canada

A Formal and Computational Account of Database Transactions

We address the problem of formalizing the evolution of a database under the effect of an arbitrary sequence of update transactions. We do so by appealing to a first order representation language called the situation calculus, which is a standard approach in artificial intelligence to the formalization of planning problems. We formalize database transactions in exactly the same way as actions in the artificial intelligence planning domain. This leads to a database version of the frame problem in artificial intelligence. We provide a solution to the frame problem for a special, but substantial, class of update transactions. Using the axioms corresponding to this solution, we provide procedures for determining whether a given sequence of update transactions is legal, and for query evaluation in an updated database. These procedures have the nice property that they appeal to theorem-proving only with respect to the initial database state.

We next address the problem of proving properties true in all states of the database. It turns out that mathematical induction is required for this

task, and we formulate a number of suitable induction principles. Among those properties of database states that we wish to prove are the standard database notions of static and dynamic integrity constraints. In our setting, these emerge as inductive entailments of the database.

Finally, we describe a new logic programming language called GOLOG, whose interpreter automatically maintains an explicit representation of the dynamic world being modeled, on the basis of user supplied axioms about the preconditions and effects of actions and the initial state of the world. This allows programs to reason about the state of the world and consider the effects of various possible courses of action before committing to a particular behavior. The net effect is that programs may be written at a much higher level of abstraction than is usually possible. The language appears well suited for applications in database transaction processing, high level control of robots and industrial processes, intelligent software agents, discrete event simulation, etc. It is based on a formal theory of action specified in an extended version of the situation calculus. A prototype implementation in Prolog has been developed.

The above material was drawn from the following papers:

R. Reiter. On specifying database updates. J. Logic Programming, 25, Oct. 1995.

H. Levesque, R. Reiter, Y. Lesperance, F. Lin and R. Scherl. GOLOG: a logic programming language for dynamic domains. J. Logic Programming. To appear.

R. Reiter. Proving properties of states in the situation calculus. Artificial Intelligence 64, 1993, pp. 337–351.

Peter Z. Revesz
<http://www.cse.unl.edu/~revesz>
<revesz@tamana.unl.ed>

University of Nebraska
Dept. of Computer Science
106 Ferguson Hall
Lincoln NE 68506-0115, USA

Model-Theoretic Minimal Change Operators for Constraint Databases

We distinguish among three types of database change operators:

1. *Revision –which is used to add some more reliable information to a current knowledgebase storing information about a static world.*

2. *Update –which is used to add some more up-to-date information to a knowledgebase storing information about a dynamic world where objects and relationships among objects change with time.*

3. *Arbitration –which is used to add together information from two or more sources that are similarly reliable and up-to-date to form a consensus knowledgebase.*

Revision is characterized by axioms given in Alchorron, Gardenfors, and Makinson (1985), update by axioms given in Katsuno and Mendelzon (1991), and arbitration by axioms given in Revesz (1993). The set of revision, update, and arbitration operators are pairwise disjoint, and each type of operators accomplishes a model-theoretic minimal change in a distinct way.

Unfortunately, previous examples of model-theoretic change operators such as the update operator of Grahne, Mendelzon, and Revesz (1992) had the following limitations: (a) not dealing with the frame and ramification problems in artificial intelligence, (b) limiting quantifier scopes to the active domain and (c) not keeping histories. The paper shows how these problems can be solved by considering integrity constraints on knowledgebases and by considering constraint query languages as the description language for the new information to be added to the knowledgebase.

Domenico Saccà
<sacca@unical.it>

DEIS Department
Università della Calabria
87030 Rende, Italy

Nondeterminism in database languages is mandatory

Although both theory and practice in logic database query languages have achieved many important results, yet logic query languages are far to be widely used and, besides, DATALOG is not sold in software stores. This unsatisfactory situation is determined by many factors. Our claim is that one of such factors is that logic query languages are required to be both 'generic' and 'deterministic' and such two properties are mutually exclusive. Indeed, a non-boolean query (corresponding to a search problem) has in general several equivalent solutions so that one of them is to be singled out in order to preserve non-determinism. But then the query looses genericity as low-level, implementation-dependent (e.g., some ordering on the data) details must be introduced to distinguish one solution from the others. Non-deterministic query languages represents a simple solution to the

problem. Moreover, there are results demonstrating that non-deterministic queries cannot be implemented (i.e., refined) using deterministic queries, implying that non-determinism is intrinsic to the genericity constraint characterizing queries. Further on, we show that the large expressive power of non-determinism can be disciplined so that polynomial time behavior is guaranteed with polynomial time problem while higher costs are enabled only when they are required by the problem at the hand. Finally, our belief is that, in order to avoid the mentioned problems of logic query languages, on-going research on logic update languages should take into account a certain amount of non-determinism.

Ernest Teniente
<teniente@lsi.upc.es>

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo, 5
E-08028 Barcelona - Catalonia

Classification and Specification of Update Problems

everal problems may arise when updating a deductive database. During last years much research has been devoted to different database update problems like view updating, integrity constraints checking, materialized view maintenance, integrity constraints maintenance or condition monitoring.

Up to now, the general approach of the research related to these problems has been to provide specific methods for solving particular problems. However, most of these methods are explicitly or implicitly based on a set of rules that define the changes that occur in a transition from an old state of a database to a new one. Therefore, these rules provide the basis of a framework for classifying and specifying these problems. We propose to use the event rules, which explicitly define the insertions and deletions induced by an update, for such a basis.

We define two interpretations of the event rules: the upward interpretation and the downward one. The upward interpretation defines the changes on derived predicates induced by a transaction which consists of a set of changes on base facts. On the other hand, the downward interpretation defines the changes on base predicates needed to satisfy a given set of changes on derived predicates.

For classifying and specifying database updating problems in terms of these interpretations, we need to endow a derived predicate with a concrete semantics. An integrity constraint Ic, a view View (materialized or not) and a con-

dition to be monitored Cond can be expressed as a derived predicate. Thus, the upward and downward interpretation of the event rules corresponding to Ic, View and Cond allow us to classify and to specify the deductive database updating problems that have been addressed in the past years and to identify also some other problems that, to our knowledge, have not been addressed up to now.

By considering only a unique set of rules for specifying all these problems, we want to show that it is possible to provide general methods able to deal with all these problems as a whole. Therefore, we could uniformly integrate view updating, materialized view maintenance, integrity constraints checking, integrity constraints maintenance, condition monitoring and other deductive database updating problems into an update processing system.

Finally, it is important to point out that we are not proposing a new method for change computation. What we propose is to use the event rules and their interpretations as a common framework for classifying and specifying deductive database updating problems. A particular implementation of these interpretations would produce a particular method for change computation.

This is a joint work with Toni Urpí

Toni Urpí
<urpi@lsi.upc.es>

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo, 5
E-08028 Barcelona - Catalonia

Classification and Specification of Update Problems

See the entry for Ernest Teniente

Helmut Veith
<http://www.dbai.tuwien.ac.at/staff/veith.html>
<veith@dbai.tuwien.ac.at>

Information Systems Department
Technical University of Vienna
Paniglgasse 16
A-1040 Wien, Austria

Logic Programming: Modularity and Revisions

In the programming methodology of logic programming it is important to have the possiblity of building program modules, which can be used to develop a complex program. These program modules act like subprograms, and are used to compute intermediate results by the main program, which calls the subprograms on specific input data.

In our talk we argue that logic subprograms can be seen as a certain kind of generalized quantifiers*, and propose a semantics based on stable models which allows for reusable logic subprograms.*

*We generalize the notion of dependency graph to the case of modular programs, and identify syntactic properties of modular logic programs akin to stratification (*semistratified, call recursion free, call independent programs*).*

For those systems, we investigate the role of different call modalities, the nesting depth of modules, and the notion of monotone subprograms*.*

Building on results from finite model theory, we show normal forms for several of those classes, and relate the syntactic classes to well-known complexity classes from the polynomial hierarchy. As an immediate consequence, we see that disjunctive logic programming is just as expressive as modular programming with nesting depth 2*.*

A natural application stems from revision programming: In a recent series of papers by V.W. Marek, T.C. Przymusiński, M. Truszczyński and H. Turner, a concept of revision programming which is closely related to the stable model semantics was developed. We investigate the use of revision programs as subprograms, as well as the evaluation of subjunctive queries in the proposed framework of modular logic programming.

This is a joint work with Thomas Eiter and Georg Gottlob.

Carl-Alexander Wichert
<http://www.uni-passau.de/~wichert/>
<wichert@fmi.uni-passau.de>

Universität Passau
Fakultät für Mathematik und Informatik
94030 Passau
Germany

Logical Specification of Bulk Updates and Sequential Updates

We describe a rule-based update specification language which generalizes the classical deductive database languages. In our framework, procedural update methods can be combined with set-oriented updates. All conventional database update operations are supported.

Update operations can be hierarchically constructed starting from (predefined) basic update operations, like insert tuple *and* delete tuple*. The complex update operations are defined by update rules having as subgoals retrieval operations, i.e. conventional literals, basic update requests, or (already defined) complex update atoms. There are three major composition constructs: sequential composition (sequential AND), parallel composition (parallel AND), and bulk update quantification. An update rule can be regarded as a (partial) definition of the transaction named by the head predicate and parameterized by the head variables.*

The main contribution of this paper is the definition of a semantics of set-oriented updates (bulk updates) that is compatible with the sequential and parallel composition of update operations. Basic updates are not executed at derivation time. Instead, sets of basic update requests are computed for each resolution path. Each composition construct determines how the update request sets corresponding to its operands have to be merged. For bulk updates the merging is essentially performed by an aggregation over success paths. Bulk updates can be regarded as a special application of parallel composition. Due to sequential composition there may exist different intermediate states which are handled by a suitable form of hypothetical reasoning.

After having collected the update requests in the "derivation phase" described above, the final update request sets can be executed on the extensional database in a subsequent "materialization phase". The concept of update request sets allows to evaluate update programs not only top-down but also

bottom-up as shown in the companion paper by the same authors.

This is a joint work with Burkhard Freitag

Carl-Alexander Wichert
<http://www.uni-passau.de/~wichert/>
<wichert@fmi.uni-passau.de>

Universität Passau
Fakultät für Mathematik und Informatik
94030 Passau
Germany

Using Hypothetical States to Implement Bulk Updates and Sequential Updates

See the entry for Burkhart Freitag

Kazumasa Yokota
<http://banjo.kuis.kyoto-u.ac.jp/~yokota/E/>
<yokota@kuis.kyoto-u.ac.jp>

Department of Information Science
Kyoto University
Sakyo, Kyoto 606-01, JAPAN

Hypothetical Query Processing for Workflow Management

Our contributions in this paper are a proposal of a workflow data model and application of hypothetical query processing techniques to this application.

In the Japanese FGCS project, we have developed a DOOD (deductive object-oriented database) language and its system, Quixote, and a multiagent based problem solver, Helios. Especially, hypothetical query processing and abductive reasoning in Quixote are very effective for many knowledge information processing applications such as legal reasoning and natural language processing.

In this paper, we take workflow management as an application of Quixote and Helios in distributed environments. Our model of a workflow database consists of a production system and a set of definite clauses, which control dynamically structured works by forward and backward reasoning. Differently from conventional approaches to workflow databases, workflows are formally defined and easy to reuse by their generalization hierarchy. Further, hypothetical query processing in workflow management contributes to constructing hypothetical workflows, personalizing multiple workflows, and discovering

bottlenecked works.

This is a joint work with Takeo Kunishima and Hideyuki Nakanishi

Carlo Zaniolo
<http://www.cs.ucla.edu/csd/>
<zaniolo@cs.ucla.edu>

Department of Computer Science
University of California
Los Angeles, CA 90024, USA

A Minimalist's Approach to the Meaning of Database Updates and Active Rules

We pursue a simple framework for modeling database updates and Event-Condition-Action rules of Active Databases. The main idea is to treat the database history as the extensional database and view the current database as a set of intentional relations derived via frame-axioms rules.

Reasoning on histories can be accomplished using $Datalog_{1S}$ programs, where active rules are viewed as deductive rules defining the next state in history from the current state in history. This approach yields programs that always have stable model semantics. The existence of stable models is in fact guaranteed by the very syntactic structure of the programs, which obey a strict form of local stratification called XY-stratification. XY-stratified programs are programs that become stratified once the temporal arguments in their predicates are instantiated. The use of non-deterministic choice constructs to model a non-deterministic firing of rules yields program that are stratified modulo choice, and thus posses multiple total stable models.

The regular structure of the resulting programs facilitate the analysis of their abstract properties and their efficient bottom-up computation.

We show that different active rule semantics, including immediate, deferred and durable-change activation modes can be modeled effectively with this approach. Therefore, we obtain a unified and simplified framework for reasoning with non-monotonic knowledge, non-deterministic constructs, database updates, active rules and a database histories.

Carlo Zaniolo
<http://www.cs.ucla.edu/csd/>
<zaniolo@cs.ucla.edu>

Department of Computer Science
University of California
Los Angeles, CA 90024, USA

Languages for Event-Based Reasoning in Active Databases and Time Series

A powerful feature of next-generation active databases is support for rules triggered by complex patterns of composite temporal events. The languages used to describe composite events provide powerful primitives for event-based temporal reasoning. In fact, with one important exception, their expressive power is comparable to that of the sophisticated languages offered by Time Series Management Systems (TSMS), which are extensively used for temporal data analysis and knowledge discovery. The important exception pertains to temporal aggregation, for which current active database prototypes offer only minimal support, if any. An additional problem with composite event rule languages is their lack of a common semantic basis, since each language uses a different approach.

In this paper, we introduce the language TREPL which addresses both previous problems using a logic-based semantics. The TREPL prototype, under development at UCLA, offers rich primitives for temporal aggregation. In fact, TREPL's power in dealing with temporal aggregates exceeds that of languages such as ODE, Snoop and SAMOS, and is comparable to that of TSMS languages.

Furthermore, TREPL demonstrates a rigorous and general approach to the definition of the semantics for active rule languages with composite events. Thus, the meaning of TREPL rules is defined by their mapping into $Datalog_{1S}$ rules, whose formal semantics then characterizes the behavior of active rules with composite event expressions. This approach, which is also applicable to languages such as ODE, Snoop and SAMOS, can be naturally extended to temporal aggregates, including user-defined ones.

This is a joint work with I. Motakis