Klaus-Dieter Schewe
Bernhard Thalheim (Eds.)

Semantics in Data and Knowledge Bases

LNCS 4925

Third International Workshop, SDKB 2008
Nantes, France, March 2008
Revised Selected Papers

Lecture Notes in Computer Science 4925

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Klaus-Dieter Schewe Bernhard Thalheim (Eds.)

Semantics in Data and Knowledge Bases

Third International Workshop, SDKB 2008
Nantes, France, March 29, 2008
Revised Selected Papers

Springer

Volume Editors

Klaus-Dieter Schewe
Information Science Research Centre
20A Manapouri Cr, Palmerston North 4410, New Zealand
E-mail: kdschewe@acm.org

Bernhard Thalheim
Christian-Albrechts-University of Kiel
Institute of Computer Science and Applied Mathematics
Olshausenstr. 40, 24098 Kiel, Germany
E-mail: thalheim@is.informatik.uni-kiel.de

Library of Congress Control Number: 2008938654

CR Subject Classification (1998): H.2, H.4, H.3, C.2.4, F.4.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-88593-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-88593-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12549405 06/3180 5 4 3 2 1 0

Preface

This volume comprises selected papers of the Third International Workshop on Semantics in Data and Knowledge Bases, which was collacated with EDBT 2008 and was organized in Nantes on March 29, 2008. The first two workshops "Semantics in Databases" took place in Rež, Czech Republic in 1995 and Dagstuhl, Germany, 2001. The workshops have had post-proceedings of selected papers given at the workshop. We invited the best papers of the workshop to submit a revised version of their paper. These revisions have been reviewed for the final proceedings. The proceedings of the first two workshops were published by Springer in the LNCS series, volumes 1358 and 2582. The SDKB 2008 workshop call for papers led to 19 submissions, which were reviewed by 4 reviewers.

We selected six of the papers given at the SDKB 2008 workshop. We additionally invited four papers that round up the proceedings. Furthermore, we added a survey on the state of the art in the field.

The SDKB workshop series tries to bring together researchers in the areas of data and knowledge bases who work on aspects of semantics. In particular, the workshop presents original contributions demonstrating the use of logic, discrete mathematics, combinatorics, domain theory and other mathematical theories of semantics for database and knowledge bases, computational linguistics and semiotics, and information and knowledge-based systems.

Topics of research papers are concentrated around the following research topics:

Formal models for data and knowledge bases
Integrity constraints maintenance and dependency theory
Formal methods for data and knowledge base design
Reasoning about data and knowledge base dynamics
Adaptivity for personalized data and knowledge bases
View-centered data- and knowledge-intensive systems
Information integration in data and knowledge bases
Knowledge discovery in data and knowledge bases
Validation and verification of data and knowledge base designs
Formal linguistics for data and knowledge bases
Logical and mathematical foundations of semantics
Semantics in data- and knowledge-intensive applications

We want to thank the members of our Program Committee for their detailed reviews and for the support in the second round of reviewing revised papers. We are thankful to the EDBT organizers for the environment and the organization

of the workshop. Our thanks go especially to Laurent Amsaleg, Elisabeth Lebret, Markus Kirchberg, and René Noack for their support of the organization.

July 2008

Klaus-Dieter Schewe
Bernhard Thalheim

Organization

Program Co-chairs

Klaus-Dieter Schewe	Information Science Research Centre, New Zealand
Bernhard Thalheim	Christian Albrechts University of Kiel, Germany

Program Committee

Catriel Beeri	Israel
Stefan Brass	Germany
Andrea Cali	Italy
Lois Delcambre	USA
Thomas Eiter	Austria
Guido Governatori	Australia
Marc Gyssens	Belgium
Roland Hausser	Germany
Stephen Hegner	Sweden
Gabriele Kern-Isberner	Germany
Markus Kirchberg	Singapore
Nicola Leone	Italy
Mark Levene	UK
Sebastian Link	New Zealand
Hui Ma	New Zealand
Wai Yin Mok	USA
Wilfred Ng	Hong Kong
Peter Revesz	USA
Attila Sali	Hungary
Dietmar Seipel	Germany
Leticia Tanca	Italy
Rodney Topor	Australia
Riccardo Torlone	Italy

Additional Reviewers

E. Asarin	D. Beauquier	P. Caspi

Organization Committee

Wolf Zimmermann	Martin Luther University Halle-Wittenberg
Rene Noack	Martin Luther University Halle-Wittenberg
Michael Schaarschmidt	Martin Luther University Halle-Wittenberg
Thomas Kobienia	Cottbus University of Technology

Sponsoring Institutions of SDKB 2008

Martin Luther University Halle-Wittenberg, Germany
Microsoft Research, Redmond, WA, USA
Stiftung Leucorea, Germany
Winzervereinigung Freyburg-Unstrut eG, Germany

Table of Contents

Introduction

Invited Papers

Research Papers

Semantics in Data and Knowledge Bases

Klaus-Dieter Schewe[1] and Bernhard Thalheim[2]

[1] Information Science Research Centre,
20A Manapouri Cr, Palmerston North 4410, New Zealand
[2] Christian Albrechts University Kiel, Department of Computer Science,
Olshausenstr. 40, D-24098 Kiel, Germany

kdschewe@acm.org, thalheim@is.informatik.uni-kiel.de

1 Semantics

Semantics is the study of meaning, i.e. how meaning is constructed, interpreted, clarified, obscured, illustrated, simplified, negotiated, contradicted and paraphrased [Wan87]. It has been treated differently in the scientific community, e.g., in the area of knowledge bases and by database users.

- The scientific community prefers the treatment of 'always valid' semantics based on the mathematical logic. A constraint is valid if this is the case in any correct database.
- Database modellers often use a 'strong' semantics for several classes of constraints. Cardinality constraints are based on the requirement that databases exist for both cases, for the minimal and for the maximal case.
- Database mining is based on a 'may be valid' semantics. A constraint is considered to be a candidate for a valid formula.
- Users usually use a weak 'in most cases valid' semantics. They consider a constraint to be valid if this is the usual case.
- Different groups of users use an 'epistemic' semantics. For each of the group its set of constraints is valid in their data. Different sets of constraints can even contradict.

Semantics is currently one of the most overused notions in modern computer science literature. Its understanding spans from synonyms for structuring or synonyms for structuring on the basis of words to precise defined semantics. This partial misuse results in a mismatch of languages, in neglecting formal foundations, and in brute-force definitions of the meaning of syntactic constructions.

1.1 Variety of Notions for Semantics

We are thus interested in a clarification of the notion of semantics. The notion of semantics is used in different meanings:

Lexical semantics is the study of how and what the words in a language denote. It uses a theory of classification and decomposition of word meaning and of association of words via relationships such as hyponomy, synonymy, troponymy, hypernymy, and antonymy. Word fields are the main notion for lexical semantics. An ontology

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 1–25, 2008.

is typically based on word fields in a rudimentary form and on selected associations among words. In general, an ontology provides a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed application systems. This meaning is the basis for "Semantic" Web. It uses a rudimentary form of word semantics for meta-characterisation.

Grammatical semantics uses categories such as 'noun' for things, 'verb' for actions and 'adjectives' for descriptions. Categories are enhanced by grammatical elements. Grammatical rules describe which syntactic expressions are well-formed. It is often assumed that the syntactic properties of words are determined by their meaning. Combinatorial semantics is a specific form of grammatical semantics.

Statistical semantics bases the meaning on co-occurrence of words, on pattern of words in phrases, and on frequency and order of recurrence. Information retrieval applications use this co-occurrence pattern for for similarity detection, keyword extraction, measuring cohesiveness in texts, discovering different senses of the word, and for analysing and mining.

Logical semantics is based on a relation between the formal language of logics defined on some alphabet or signature from one side and the set of structures or worlds defined on the same signature from the other side. The relation serves as the means for realising the assignment and characterises at the same time the truth of expressions via their interpretation in the set of structures. The truth value of an expression is a statement at a meta-language level.

Prototype semantics bases meaning on users evolving experience. It is thus culture and community dependent. The semantical structure used for definition of the meaning consists of different elements which have unequal status. This leads to a graded notion and varying structuring within semantical structures. Some elements are better for clarification of the concepts. Elements are are mainly possible contributors to the meaning. Semantical structures may be layered into basic level ones and combined ones.

Program and dynamic semantics is based on the semantic memory, i.e. the memory of meanings, understandings, and other concept-based knowledge unrelated to specific experiences of agents. Dynamic semantic structures may be represented by generic [BST06] or abstract structures (e.g. semantic networks, associative or feature or statistical models) that apply to a wide variety of experimental objects. It overcomes limitations of the nativist view on semantics and explicitly considers evolution of meaning depending on a state of a reference system such as the computer and depending on the context and agents. Database semantics is based on expression-linear definition of meaning and allows to reduce the meaning to those components that are necessary for definition of the meaning. Dynamic semantics supports semantic under-specification and transfer of meaning by injection of context and agent opinion.

Semantics can be based

- on constructors and compositions in the sense of Frege that the interpretation can be defined by the interpretation of a constructor and an interpretation of constituents of complex expressions and the interpretation of basic is provided by assumptions,

- on context in the sense of reduction logics by assigning a set of rules for expansion of expressions to expressions for a meaning is defined and by defining agents with roles such as hearer and speaker that provide an understanding in their worlds, e.g., reduction semantics used in database semantics for reduction of components of sentences to proplets, and
- on transformations to other languages for which a semantics is well-defined and by transformation rules which meaning is well-understood and and concise.

The principle of compositionality postulates that the meaning of an expression is determined by its structure and its constituents. This principle is productive in the sense that meaning can be derived without any explicit definition in advance and is systematical in the sense that there are definite and predictable pattern among expressions. Expressions are referentially transparent since they can be replaced or instantiated by values without changing the expression. Queries are referentially transparent whereas commands in non-functional languages may change the environment and are context-dependent. Compositionality is one of the main conditions for axiomatisability of logics. Compositionality cannot be maintained within context-dependent or time-dependent structures.

Considering more complex type systems the treatment of missing values adds another dimension to semantics. For instance, in the entity-relationship model [Tha00a] entity types can be defined on set semantics and relationship types can be based on pointer (or reference) semantics enabling thus the developer to reason on 'missing links'. This difference in the treatment of semantics has led to some of the problems discussed for view integration. We therefore established this workshop seria together with the FoIKS conferences in order to maintain the knowledge on semantics in both fields.

1.2 The Semiotics Background of Semantics and Pragmatism

Database and knowledge base theory use languages. Therefore, we are bound to the conceptions of the language, the expressivity of the language, and the methodology for language utilisation. The Sapir-Whorf observation [Who80] postulates that developers skilled in a certain language may not have a (deep) understanding of some concepts of other languages.

Semantics is the study and knowledge of meaning in languages. It is a central component of semiotics. Modern semantics [Hau01] is based on a notion of meaning and reference, on a notion of semantic spaces, on a construction of semantic structures such as semantic fields, on a notion of semantic relations among constructions, on semantic components, and on a notion of prosodic, grammatical, pragmatical, social and propositional meaning of constructions. *Semiotics* distinguishes between syntactics (concerned with the syntax, i.e., the construction of the language), semantics (concerned with the interpretation of the words of the language), and pragmatics (concerned with the meaning of words to the user). Most languages used in computer science have a well-defined syntax. Their semantics is provided by implementations which are often not made public and their pragmatics is left to experimentation by the user. This 'banana' or 'potato' principle leads to very different utilization and does not allow consistent use by groups of developers and over time. Each step is based on a specification language that has its syntax, semantics, and pragmatics.

Syntactics: Inductive specification of structures uses a set of base types or a vocabulary or an alphabet, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. Structural recursion is the main specification vehicle. Constructors may be defined on the basis of grammatical rules.

Semantics: Specification of admissible structures on the basis of static (integrity) constraints describes those structures which are considered to be legal. If structural recursion is used then a variant of hierarchical first-order predicate logics may be used for description of constraints.

Pragmatics: Description of context and intension is based either on explicit reference to the model, to tasks, to e policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

These main branches of semiotics cannot be entirely layered and are intertwined. The meaning of a word depends on its use [Wit58] and its users. We often assume however that syntax is given first. We should distinguish between expression meaning, statement meaning, and context or utterance meaning. The first one is designates the semantic properties an expression possesses merely by the virtue of being well formed and is based on the propositional content of the expression, i.e. propositions or semantical units it contains, on the interrogative, imperative, and/or expressive meaning, and finally on features of formal distinction. A statement postulates that some state of affairs holds. It has a truth value. The statement meaning has two parts: an element of assertion and something that is asserted. What is asserted is called proposition. The proposition is an expression that can be believed,doubted, or denied or is either true or false. The simplest type of proposition consists of an argument and a predicate. The context meaning depends on the intention of the sender and the relationship of the sender with the speaker and is based on auxiliary expressions shared between sender and receiver.

We also may distinguish between the existential approach to meaning based on a correlation of expressions in a language with aspects in the world. The intentional approach associates some kind of representation with concepts as the main constituents of the sense and depends on the cultural context. Semantical units or propositions are interrelated by entailment. Entailment is different from material implication and relates propositions by forward propagation of truth and backward propagation of falsity. Propositions can be contraries, contradictories, or independent. They may belong to a category or genre of expression, are given in a certain style or manner, are often based on stereotypical norms of expression, depend on ideas and values that are employed to justify, support or guide the expression, reflect aspects of culture or social order, are shaped according to the community that uses them, and are configured by theories or paradigms.

Pragmatism means a practical approach to problems or affairs. According to Webster [Web91] pragmatism is a 'balance between principles and practical usage'. Thus, it is a way of considering things. Pragmatism may be based on methodologies, e.g., database design methodologies. For instance, the co-design methodology of database development [Tha00a] enables in consistent design of structuring, functionality, interactivity and distribution of information systems. The constituents of the methodology are, however, constructs of the design language and constructors used to construct complex

construction. Constructs we use are the elements of the extended entity-relationship models [Tha00a]: attribute types, entity types, relationship types of arbitrary order, and cluster types. Constructors used for defining more complex types are: (Cartesian) product constructor (for associating types) and list, tree set and bag constructors (for constructing groups or collections of types)[Tha00b]. The way of considering things or the practical usage has been investigated in less depth. Schemata have a certain *internal structuring* or *meta-structuring*. Thus, pragmatism of modelling should also be based on principle to handle the meta-structuring within the schemata.

1.3 Formal Semantics in General

Formal semantics is typically based on a system that allows to reason on properties of systems. It aims in describing syntactic and semantic types or elements that are used as well as the meaning function that maps syntactic types to semantic types. There are several forms of formal semantics: operational, denotational, axiomatic, transformational, algebraic, macro-expansion, and grammar semantics.

Formal semantics typically assume *compositionality*. The principle of compositionality is not necessary and may contradict applications. Compositionality assumes that the elements are independent from each other. For instance, the classical XOR-connective $\neg(\alpha \wedge \beta)$ defines incompatibility of α and β. It thus provides a context for subexpressions. Idioms, non compounds, active zones and complex categories cannot be based on compositionality. If compositionality is assumed we may distinguish between the additive mode where the meaning of components is added for the constructor and the interactive mode where the meaning of at least one components is radically modified. Formal semantics mainly uses the additive mode. Natural, lexical, prototype, statistical and partially program semantics use the interactive mode. The meaning of a complex expression can either be of the same basic type as one of the components (endocentric) or of a different type (exocentric).

Different variants of semantics is used in computer science [Bak95, BS03, Gun92], [Mos92, Rey99, Ten97, Tha00a]. A formal semantics [Cry87, Sch71, Sch72, Ste73] is typically given

- by an *interpreter* that maps syntactic types to semantic types,
- by a *context abstraction* that is based on an aggregation of values which remain fixed in certain temporal and spatial intervals,
- by *states* that provide a means of representing changes over time and space,
- by an *configuration* that is based on an association of contexts and states,
- by an *interpretation function* that yields state results based on certain computation,
- by an *evaluation function* that yield some value results for the syntactic types, and
- by an *elaboration function* that yield both state and some other value results.

These mapping are often given in a canonical setting, i.e. interpreters are defined on signatures, context abstraction is neglected due to use of the most general types, states are based on mathematical structures, configurations are fixed, and interpretation functions are given in a canonical setting. The evaluation function can be freely configured.

The elaboration function is used for the definition of aggregates such as the quantifiers. Mathematical logics and formal semantics in computer science is defined by such canonical setting.

In computer science we may distinguish between static semantics that lays down what it means for terms to be well defined (beyond syntactic criteria) and dynamic semantics that determines the meaning of terms. Dynamic semantics is mainly defined through operational semantics either via small-step operational semantics (structural operational semantics or reduction semantics) or big-step operational semantics (natural semantics or evaluation semantics), or through denotational semantics or through axiomatic semantics. Additionally, transformational and algebraic approaches are used.

These seven mappings are the basis for a variety of definitions of semantics. Formal semantics does not oblige any restriction by states or context. It uses a strict matching between the syntactical language and the world of semantical structures either by strict association or by embedding of expressions in the syntactical language $\mathcal{L}$ into the language $\mathcal{L}'$ in the world of semantical structures. In the opposite, matching is only partial for natural logics. We may for instance use different kinds of mappings for different logics:

mappings	logics	closed world logics	logics on finite worlds	logics on natural worlds
matching of syntactic language $\mathcal{L}$ and semantic structure worlds $\mathcal{W}$ on signature τ	exact or coincidence embedding	exact	exact or coincidence embedding	partial depending on interest and meaning in use
considering context	no	no	no	depending on use and user
considering states	any	any	only finite structures	states in scope
restricting states and context	no	no	no	depending on interest and demand
interpretation for alphabets	exact for alphabet	exact	potentially restricted	multiple interpretations
evaluation of variables	full	full	full	partial evaluation
elaboration	full	negation as failure	derivable structures	extrapolation

This table shows the variety of approaches that might be used for the definition of semantics. Application models restrict languages to those parts that are currently of interest. The context also considers enterprise and social models. Integrity constraints used in database models are not interpreted for specific values such as null values. Interpretation of expressions is restricted to meaningful expressions in finite worlds, e.g. the tuple-relational calculus in relational database theory only defines the meaning of those expressions that map finite structures to finite structures.

Formal semantics may be understood as the semantics of formal languages. Model-theoretic semantics is widely used in computer science, e.g. for semantics of databases. It defines the meaning by recursive mapping to some predefined mathematical structures and by assigning propositions to truth values. Proof-theoretic semantics is widely used in

artificial intelligence and for knowledge bases. It associates the meaning of propositions with the roles that they play in inferences. Truth-value semantics assigns a truth value to expressions without reference to structures. Game and probabilistic semantics are other kinds of formal semantics.

1.4 Semantics of Mathematical Logics as One Kind of Semantics

Mathematical logics uses a canonical way of associating syntactic and semantic types. Additionally, the semantic type and the syntactic type have the same signature. The expressions of syntactic types are inductively constructed starting with some basic expressions of certain construct by application of expressions of some other construct. For instance, we may start with truth values and variables. Terms and formulas are then based on these basic expressions.

The semantic type LOG of first-order mathematical logics uses the two truth values $true$ and $false$, canonical connectives $\wedge, \vee$ and $\neg$ and the quantifiers $\forall$ and$\exists$.

The interpreter is given by a canonical mapping. Context abstraction is defined by the trivial context and thus not considered. Most logics do not consider changing states. The interpretation function is an inductive function that is based on interpretation of basic expressions and follows the construction of expressions. The evaluation and elaboration functions allow to consider different value assignments to an expression.

The correspondence mapping in logics can be combined with additional constructs such as validity, satisfiability, and conclusions. A expression of the construct 'formula' is considered to be true if it is mapped to the truth value $true$ from LOG. A expression of a syntactic type follows from a set of expressions if the truth value $true$ is assigned to the expression whenever this truth value has been assigned to the set of expressions.

This understanding of logics in general may be defined on transparent intensional logics [DM00]. The association of syntactic types and semantic types is typically rather restricted. In some case, we might broaden this specific form of semantics.

Classical predicate logics is based on monotone reasoning. The derivation operator $\vdash_\Gamma$ for a calculus Γ and the conclusion operators $\models$ for the inheritance of validity in structures are monotone, i.e for sets of formulas Σ, Σ' and a formula α we may conclude that $\Sigma \cup \Sigma' \vdash_\Gamma \alpha$ or $\sigma \cup \Sigma' \models \alpha$ if $\Sigma \vdash_\Gamma \alpha$ or $\sigma \models \alpha$, correspondingly. Operators $\mathcal{A}_\Psi$ define a closure for a set-formula relation Ψ on $\mathcal{L}$, e.g., $Cons(X) = \{\alpha \in \mathcal{L} \mid X \models \alpha\}$ and $Deriv(X) = \{\alpha \in \mathcal{L} \mid Ax \cup X \vdash_\Gamma \alpha\}$ for a set of axioms Ax.
The first main theorem of monotone reasoning states that $\Sigma \cup Ax \vdash_\Gamma \alpha$ if and only if $\Sigma \models \alpha$ for any set $\Sigma \subseteq \mathcal{L}$ and for any formula $\alpha \in \mathcal{L}$ (Completeness and soundness).

Useful properties for relationships Ψ and their closure operators $\mathcal{A}_\Psi$ are

1. Reflexivity: $\Sigma \subseteq \mathcal{A}_\Psi(\Sigma)$.
2. Monotonicity: If $X \subseteq Y$ then $\mathcal{A}_\Psi(X) \subseteq \mathcal{A}_\Psi(Y)$.
3. Closure: $\mathcal{A}_\Psi(\mathcal{A}_\Psi(\Sigma)) = \mathcal{A}_\Psi(\Sigma)$.
4. Compactness: If $\alpha \in \mathcal{A}_\Psi(X)$ then there exists a finite set $X' \subseteq X$ such that $\alpha \in \mathcal{A}_\Psi(X')$.
5. Inference property: If $\alpha \rightarrow \beta \in \mathcal{A}_\Psi(X)$ then $\beta \in \mathcal{A}_\Psi(X \cup \{\alpha\})$.
6. Deduction property: If $\alpha \in \mathcal{A}_\Psi(X \cup \{\beta\})$ and β is closed then $\beta \rightarrow \alpha \in \mathcal{A}_\Psi(X)$.
7. Generalization invariance: Let $Gen(\Sigma)$ denote the set of generalization of formulas from Σ . $\mathcal{A}_\Psi(Gen(X)) = \mathcal{A}_\Psi(X)$.

8. Transitivity: If $\alpha \in \mathcal{A}_\Psi(\Sigma)$ and $\beta \in \mathcal{A}_\Psi(\Sigma \cup \{\alpha\})$ then $\beta \in \mathcal{A}_\Psi(\Sigma)$.

The second main theorem of monotone reasoning states that if $\mathcal{A}_{\Psi_1}$ and $\mathcal{A}_{\Psi_2}$ are mappings from $2^{\mathcal{L}}$ into $2^{\mathcal{L}}$ with the property $\mathcal{A}_{\Psi_1}(\emptyset) = \mathcal{A}_{\Psi_2}(\emptyset)$ which fulfill the conditions 1. ... 7. then for every $\Sigma \subseteq \mathcal{L}$ $\mathcal{A}_{\Psi_1}(\Sigma) = \mathcal{A}_{\Psi_2}(\Sigma)$.
The third main theorem states that $Cons$ and $Deriv$ fulfill the conditions 1. ... 7. and $Cons(\emptyset) = Deriv(\emptyset)$.
The fourth main theorem states that for any mapping $\mathcal{A}_\Psi : 2^{\mathcal{L}} \to 2^{\mathcal{L}}$ the following assertions are equivalent:

(i). $\mathcal{A}_\Psi$ is a compact closure operator (i.e. fulfills the conditions 1.,2.,3.,4.).
(ii). There exist a system $\vdash_\Gamma$ of deduction rules on $\mathcal{L}$ such that $\mathcal{A}_\Psi \equiv \mathcal{A}_{\vdash_\Gamma}$.

The fourth theorem gives a very powerful characterisation of axiomatisability of closure operators. This characterisation covers axiomatisability of classical predicate logics and of Prolog-based predicate logics. The last one does not have the property of generalization invariance. It obeys the strong deduction property, i.e. if $\alpha \in \mathcal{A}_\Psi(X \cup \{\beta\})$ then $\beta \to \alpha \in \mathcal{A}_\Psi(X)$.

1.5 Formal Semantics in Computer Science

Computer science uses a variety of formal semantics:

Operational semantics interprets syntactic types by computational types of a possibly recursive machine may be based on varying context abstraction, and is often expressed in terms of state changes. Operational semantics is often defined on the basis of traces or sequences of states of the machine.

Denotational semantics associate mathematical functions to syntactic types, abstracts from context abstraction, and uses abstract states whose usage is based on the homomorphism principle. Denotational semantics uses partially ordered sets and their topology and thus defines Scott or Girard domains for interpretation.

Axiomatic semantics uses a calculus with axioms and proof rules that provides a mechanism for deriving correct syntactic types. Relational semantics is a special case of axiomatic semantics and is based on a specification of a start state and a set of final states. Functional semantics additionally assumes the set of final states to be singleton. Axiomatic semantics has also been defined through predicate transformers that provide a facility for backward reasoning starting with a target state.

Transformational semantics uses mappings to other syntactic types. They thus base semantics both on the (hopefully) well-defined mappings and on the semantics of the target system. Denotational semantics is a specific form of transformational semantics. It uses a mathematical formalism instead of another computer language. Categorical or functorial semantics is based on a translation to category theory.

Algebraic semantics uses a set of abstract basic syntactic systems with their semantics and a set of rules for construction of more complex systems based on these systems. Algebraic semantics can be defined through initial and final algebras or structures.

Macro-expansion semantics is based on static or dynamic inductive rewriting of syntactic types and allows to introduce abstractions such as the types of the λ calculus. Abstraction may be directly defined or based on a notion of fixed points.

Grammar semantics is a specific form of operational semantics and uses a state consisting of semantic category variables and on instantiations for atomic non-terminal syntactic types.

These semantics can be modularised or layered. For instance, action semantics uses predefined expressions (action, data, and yielders) and denotational semantics.

Semantics is often mixed with "concrete semantics" [Bjø06] that provides and 'everyday meaning' and that is mixed with 'pragmatic structures'. Often transformational semantics are used for interpretation of syntactic types. This interpretation is given in an informal way without providing a well-founded mapping and without precise semantics of the target domain. A typical example of such approach is the BPMN semantics that is based on informal description and on partial and changing mappings to BPEL types which are defined by various concrete implementations. Algebraic semantics are convenient if systems can be constructed from basic systems without limiting the construction process itself. Typical constructions of this kind are abstract data types, entity-relationship or object-relational database types, and frame types used in AI. It supports modularisation and separation of concern.

1.6 The Duality of Syntactic and Semantic Types

It is often claimed (e.g. [HR04]) that syntactic and semantic types cannot be defined. Mathematical logics uses a canonical approach to the definition of types. It starts with an enumeration of the alphabet. Words are constructed by certain rules over this alphabet. If elements of the alphabet can be categorised then we may consider similar alphabets and define a *signature* of elements of the language based on this categorisation. We may concentrate on all languages of the same signature. Computer science uses a similar approach by defining types through domains, a set of operations defined on the domains, and a set of predicates for the domains.

Formal languages have clearly defined and separated layers and use canonical techniques for construction of values of the language. We start with the enumeration of an alphabet. Some combinations of elements of these alphabets form words. Some of these words are considered to be reserved words which usage is restricted. The next layer groups words into expressions. The final layer constraints the expressions by additional conditions.

A *syntactic type* is given by one (or more) language(s), constructs, and a set of (concrete or abstract) expressions of interest.
Let $\mathcal{L}$ be a set of (supported) languages, $\mathcal{C}_L$ the set of all constructs of a certain modelling language $L \in \mathcal{L}$ and $\mathcal{S}_C$ the set of all possible states of a certain construct $C \in \mathcal{C}_L$. An expression of the syntactic type T is a triple (L, C, S) denoting a language $L \in \mathcal{L}$, a construct $C \in \mathcal{C}_L$, and an instance $S \in \mathcal{S}_C$.
Typically, one of the language is canonical. In this case we also consider a canonical set of constructs. We may omit the language and the set of constructs if we concentrate on the canonical language and set of constructs.

By a *semantic type* we understand a set of (concrete or abstract) semantic values within a certain (or some) language(s) and for a set of expressions: meanings of syntactic values. We may restrict the meaning of syntactic values to semantic values of the same (or expanded) signature. We assume one special type LOG that provides truth

values. We may also consider different variations or worlds. Let T_W a syntactic type of all worlds of interest. The set of possible worlds may also represent different versions of time and space.

The semantic type is typically defined by a state consisting of a set of individuals of certain categories, by operations and by predicates. The type LOG consists of truth values, connectives defined over these truth values, and generic quantifiers that provide a abstraction mechanism for some of its parameters to sets of values of other types. Examples of generic quantifiers are classical existence and generalisation quantifiers as well as generalised one- or two-dimensional quantifiers that are applied to pairs of expressions and describe the behaviour of components of the expressions in the whole. Typical generalised one-dimensional quantifiers are majority, many, at least and odd. Quantifiers may also be temporal ones.

In some case we might also use signatures that are completely different. One of the misconceptions of formality in computer science is the belief that only such languages are formal that uses mathematical symbols for expressions. Visual and diagrammatic languages might be formal as well.

The interpreter is based on an association of syntactic and semantic types and on an intensional function $f : \omega_{i_1} \times \omega_{i_k} \times E \to \alpha$ which maps possible worlds $(w_{i_1}, ..., w_{i_k})$ and expressions E to elements of a semantic type α. Intensional functions may be evaluated but do not reveal the internal structure of the valuation or their relationship to other intensional functions. The intensional function can also be built by an intensional construction that is used to relate valuations of the function with valuations of other first order types. The context may either be a singleton context or a set of contexts. Given a a context c, a set of intensional functions $\mathcal{F} = \{f_1, ..., f_n\}$, a logical language L and $\mathcal{T}$ a 'theory' of expressions from L that formulate the knowledge about the valuations of $\mathcal{F}$. The tuple $\mathfrak{B} = (\mathcal{F}, \mathcal{T})$ is called a concept of a syntactic type in context c. Let $\mathfrak{S}$ be a set of states. A configuration is a set of functions $i : \mathfrak{S} \to \mathcal{B}$ mapping states from $\mathfrak{S}$ to concepts from $\mathcal{B} = \{\mathfrak{B}_1, ..., \mathfrak{B}_n\}$ in the current context.

The interpretation function can now be understood as a derived function for a syntactic type and a set of semantic types. Basic expressions are mapped according to the intentional function. Elaboration functions are assigned generic quantifiers.

1.7 Micro-semantics of Wordings

Semantics often relies on pre-interpretation of some of its constructs and expressions. We may restrict the interpreter to a certain language, to a certain context and to certain states. In this case, the association of expressions to concepts becomes biased to a specific semantics. This micro-semantics is typically used with syntactic expressions such as words. The word has already a specific meaning. For instance, reserved words in programming languages are bound to a specific semantics.

This micro-semantics is often also used in applications of the so-called semantic web. Words are already provided together with a name space and a pre-interpretation. This pre-interpretation cannot be changed unless the name space is going to be changed. Communities agree on a common name space and thus introduce their ontology. The sentence meaning of XML documents is exclusively based on the words it contains and their grammatical arrangement.

Micro-semantics of wordings is also used within the theory of word fields [ST08]. Words have their own syntactical ingredients, their micro-semantics, their language usage (or language game [Wit58]) and their pragmatic application area.

Micro-semantics should not be confused with semantics. It is only a part of semantics and cannot replace semantics or used to derive semantics. It is tempting to use such micro-semantics within a culture or community. It does however not provide the complete meaning of a syntactic expression.

2 Pearls of Database Theory

Designing a database application means to specify the structure, the operations, the static semantics and the dynamic semantics. The aim is to find a full specification or at least to find a specification which leads to a database structure on which operating is simple.

Structuring of databases is based on three interleaved and dependent semiotic parts [PBGG89, Tha91]:

Syntactics: Inductive specification of database structures based on a set of base types, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. Structural recursion is the main specification vehicle.

Semantics: Specification of admissible databases on the basis of static integrity constraints describes those database states which are considered to be legal. If structural recursion is used then a variant of hierarchical first-order predicate logics may be used for description of integrity constraints.

Pragmatics: Description of context and intension is based either on explicit reference to the enterprise model, to enterprise tasks, to enterprise policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

2.1 Rigid Inductive Structures

The inductive specification of structuring is based on base types and type constructors. A base type is an algebraic structure $B = (Dom(B), Op(B), Pred(B))$ with a name, a set of values in a domain, a set of operations and a set of predicates. A class B^C on the base type is a collection of elements form $dom(B)$. Usually, B^C is required to be set. It can be list, multi-set, tree etc. Classes may be changed by applying operations. Elements of a class may be classified by the predicates.

A *type constructor* is a function from types to a new type. The constructor can be supplemented with a *selector* for retrieval (such as *Select*) and *update functions* (such as *Insert, Delete*, and *Update*) for value mapping from the new type to the component types or to the new type, with correctness criteria and rules for validation, with default rules, with one or more user representations, and with a physical representation or properties of the physical representation.

Typical constructors used for database definition are the *set*, *tuple*, *list* and *multi-set* constructors. For instance, the set type is based on another type and uses algebra

of operations such as union, intersection and complement. The retrieval function can be viewed in a straightforward manner as having a predicate parameter. The update functions such as *Insert*, *Delete* are defined as expressions of the set algebra. The user representation is using the braces $\{,\}$. The type constructors define type systems on basic data schemes, i.e. a collection of constructed data sets. In some database models, the type constructors are based on pointer semantics.

General operations on type systems can be defined by *structural recursion*. Given a types T, T' and a collection type C^T on T (e.g. set of values of type T, bags, lists) and operations such as generalized union $\cup_{C^T}$, generalized intersection $\cap_{C^T}$, and generalized empty elements $\emptyset_{C^T}$ on C^T. Given further an element h_0 on T' and two functions defined on the types $h_1 : T \to T'$
and $h_2 : T' \times T' \to T'$.
Then we define the structural recursion by insert presentation for R^C on T as follows

$srec_{h_0,h_1,h_2}(\emptyset_{C^T}) = h_0$

$srec_{h_0,h_1,h_2}(|\{\!|s|\!\}|) = h_1(s)$ for singleton collections $|\{\!|s|\!\}|$
$srec_{h_0,h_1,h_2}(|\{\!|s|\!\}| \cup_{C^T} R^C) = h_2(h_1(s), srec_{h_0,h_1,h_2}(R^C))$
iff $|\{\!|s|\!\}| \cap_{C^T} R^C = \emptyset_{C^T}$.

All operations of the relational database model and of other declarative database models can be defined by structural recursion. Structural recursion is also limited in expressive power. Nondeterministic while tuple-generating programs (or object generating programs) cannot be expressed. We observe, however, that XML together with the co-standards does not have this property.

Another very useful modelling construct is *naming*. Each concept type and each concept class has a name. These names can be used for the definition of further types.

Static integrity constraints are specified within the universe of structures defined by the structural recursion.

Observation 1
Hierarchical structuring of types leads to a generalized first-order predicate logics.

Observation 2
In general, cyclic structuring leads to non-first-order logics. Structures with abstract linking are potentially cyclic.

2.2 Static Integrity Constraints

Each structure is also based on a set of implicit model-inherent integrity constraints:

Component-construction constraints are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.

Identification constraints are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups [BT99]. We shall later see that value-representability or weak-value representability lead to controllable structuring.

Acyclicity and finiteness of structuring supports axiomatisation and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

Superficial structuring leads to representation of constraints through structures. In this case, implication of constraints is difficult to characterize.

Implicit model-inherent constraints belong to the performance and maintenance traps.

Integrity constraints can be specified based on the B(eeri-)V(ardi)-frame, i.e. by an implication with a formula for premises and a formula for the implication. BV-constraints do not lead to rigid limitation of expressibility. If structuring is hierarchic then BV-constraints can be specified within the first-order predicate logic. We may introduce a variety of different classes of integrity constraints defined:

Equality-generating constraints allow to generate for a set of objects from one class or from several classes equalities among these objects or components of these objects.

Object-generating constraints require the existence of another object set for a set of objects satisfying the premises.

A class $\mathcal{C}$ of integrity constraints is called *Hilbert-implication-closed* if it can be axiomatised by a finite set of bounded derivation rules and a finite set of axioms. It is well-known that the set of join dependencies is not Hilbert-implication-closed for relational structuring. However, an axiomatisation exists with an unbounded rule, i.e. a rule with potentially infinite premises.

Often structures include also optional components. Let us denote the set of all components of a set $\mathcal{O}$ of objects by $compon(\mathcal{O})$ and the set of all optional components of $\mathcal{O}$ by $compon^{opt}(\mathcal{O})$. Similarly we denote the set of all components used in a constraint α by $compon(\alpha)$. Validity of constraints is either based on strong semantics requiring validity for all object sets independently on whether $compon^{opt}(\mathcal{O}) \cap compon(\mathcal{O}) \neq \emptyset$ or on weak semantics requiring validity for constraints only for those object sets $\mathcal{O}$ for which $compon^{opt}(\mathcal{O}) \cap compon(\mathcal{O}) = \emptyset$. Classical validity is based on weak semantics which has a severe disadvantage:

Observation 3
Weak semantics leads to non-additivity of constraints for object sets $\mathcal{O}$ with $\mathcal{O}$ by $compon^{opt}(\mathcal{O}) \neq \emptyset$, i.e., it is not true in general that $\mathcal{O} \models \{\alpha_1, ..., \alpha_m\}$ is valid if and only if $\mathcal{O} \models \{\alpha_i\}$ for each constraint in $\{\alpha_1, ..., \alpha_m\}$.

Observation 4
Strong semantics leads to non-reflexiveness or non-transitivity of constraints for object sets $\mathcal{O}$ with $\mathcal{O}$ by $compon^{opt}(\mathcal{O}) \neq \emptyset$, i.e., $\mathcal{O} \not\models \alpha \rightarrow \alpha$ for some constraints α or the validity of $\mathcal{O} \models \alpha \rightarrow \beta$ and $\mathcal{O} \models \beta \rightarrow \gamma$ does not imply $\mathcal{O} \models \alpha \rightarrow \gamma$.

Since constraint sets may be arbitrary we might ask in which cases an axiomatisation exists. The derivation operator $\vdash_\Gamma$ of a deductive system Γ and the implication operator $\models$ may be understood as closure operators Φ, i.e.

(0) $\Phi^0(\Sigma) = \Sigma$
(i) $\Phi^{i+1}(\Sigma) = \{\alpha \in \mathcal{C} \cap \Phi(\Phi^i(\Sigma))\}$
(+) $\Phi^*(\Sigma) = lim_{i\to\infty}\Phi^i(\Sigma)$
for any subset Σ from a class $\mathcal{C}$ of constraints.

The closure operator Φ is called compact for a class $\mathcal{C}$ if the property $\alpha \in \Phi^*(\Sigma)$ implies the existence of a finite subset Σ' of Σ such that $\alpha \in \Phi^*(\Sigma')$. It is called closed of $\Phi^*(\Phi^*(\Sigma)) = \Phi^*(\Sigma)$ for any $\Sigma \subseteq \mathcal{C}$. The closure operator is called monotone if $\Phi^*(\Sigma) \subseteq \Phi^*(\Sigma \cup \Sigma')$. The operator is reflexive if $\alpha \in \Phi^*(\Sigma \cup \{\alpha\})$ for all formulas and subsets from $\mathcal{C}$.

Observation 5
The implication operator $\Phi^*_{\models}$ is reflexive, monotone, closed and compact if and only if there exists a deductive system Γ such that Φ_Γ and $\Phi_{\models}$ are equivalent. If $\Phi_{\models}$ additionally has the inference property, the deduction property and is generalization invariant then $\Phi^*_\Gamma(\emptyset) = \Phi^*_{\models}(\emptyset)$.

If the deduction property fails then the axiomatisation by a deductive system may be based on some obscure rules similar to those for the axiomatisation of PROLOG.

Constructors used for construction of more complex types are often used for convenience and representing a different structuring. A typical example is the application of the list constructor with the meaning of representing sets. In this case we must add an list-to-set axiom
$\forall t \in compon(o) \forall i,j (type(o.i) = type(o.j) = t \;\Rightarrow\; value(o.i) = value(o.j))$.
This axiom is often overseen and not considered.

Observation 6
Semantics for structures defined by the list constructor and representing set must be extended by list-to-set axiom.

Since attributes are also constructed on the basis of constructors from base types we may ask whether this construction affects the definition of constraints and the axiomatisability. This question is open for most of the constraints. In [Lin03] it has, however, shown that keys and functional dependencies have a similar treatment as in the relational case. Substructures are, however, more complex and represented by the Brouwerian algebra of subcomponents.

2.3 Application of Database Semantics to Semantical Models

The entity-relationship model has been extended to the higher-order entity-relationship model (HERM)[Tha00a]. HERM is a set-theoretic based, declarative model which objects are value-oriented. For this reason, object identifiers are omitted.

The entity-relationship model uses basic (or atomic) data types such as *INT*, *STRING*, *DATE*, etc. the null type $\perp$ (value not existing). Using type constructors for tuples, finite (multi-)sets and lists and union we construct more complex types based on standard set semantics:

$$t = l : t \mid B \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t'\} \mid \langle t' \rangle \mid [t'] \mid (a_1 : t_1) \cup \ldots \cup (a_n : t_n)$$

These types will be used to describe the domains of (nested) attributes.

Attributes allow to conceptually abstract from describing values. Associated data types provide the possible values. We use a set of *names* $\mathcal{N} = \mathcal{N}_0 \cup \mathcal{N}_c$ for attributes such as *address, street, city, name, first_name, destination, trip_course* etc. Elements from $\mathcal{N}_0$ are called atomic attributes.
Each atomic attribute is associated with a atomic type $dom(A)$.
Nested attributes are inductively constructed from simpler or atomic attributes by the iteration condition:

For already constructed nested attributes $X, X_1, ..., X_n$ and new attributes $Y, Y_1, ..., Y_m$ the sequences
$Y(X_1, ..., X_n)$, $Y\{X\}$, $Y\langle X\rangle$, $Y[X]$, $Y((Y_1 : X_1) \cup \cdots \cup (Y_n : X_n))$
are tuple-valued, set-valued, list-valued, multiset-valued and union-valued nested attributes.
Associated complex types are defined by the attribute structure. In the logical calculus below we use only tuple-valued and set-valued attributes. The calculus can similarly be extended.

For all types we use set semantics based on the basic type assignment.

Entity Types (or *level-0-types*) $E = (attr(E))$ are defined by a set $attr(E)$ of nested attributes. A subset X of attributes can be used for identification. This subset is called key of E. In this case we consider only those classes which objects can be distinguished by their values on X.

Relationship Types (or *level-$(i+1)$-types*) $R = (comp(R), attr(R))$ are defined by a tuple $comp(R)$ of component types at levels $\leq i$ with at least one level-i-type component and a set $attr(R)$ of nested attributes. We use set semantics with expanded components under the restriction that $comp(R)$ forms a key of R. Unary relationship types with $|comp(R)| = 1$ are subtypes.

Clusters (also level-i-types) $C = C_1 \oplus \cdots \oplus C_k$ are defined by a list $\langle C_1, \ldots, C_k\rangle$ of entity or relationship types or clusters (components). The maximal level of components defines level i. We set semantics (union) or equivalent pointer semantics.

Corresponding classes of a type T are denoted by T^C. $\mathcal{R}(T)$ is the set of all classes of T. Basic type assignment is equivalent to pointer semantics with *value representability*.

The usual graphical representation of the extended ER model is a labelled directed acyclic graph. Entity types are denoted graphically by rectangles. Relationship types are graphically represented by diamonds with arrows to their components. Attributes are denoted by strings and attached to the types by lines. Key components are underlined.

A HERM scheme S is a set $\{R_1, ...R_m\}$ of types of level $0, ..., k$ which is closed, i.e. each set element has either no components (entity type) or only components from $\{R_1, ...R_m\}$.

Based on the construction principles of the extended ER model we can introduce the HERM algebra [Tha00a]. In general, for a HERM scheme S the set $Rec(S)$ of all expressions definable by structural recursion can be defined.

Since HERM database schemes are acyclic and types are strictly hierarchical we can construct a many-sorted logical language by generalizing the concept of variables.

Given a HERM scheme S. Let $\mathcal{N}_S$ the set of all names used in S including type names. A sort is defined for each name from $\mathcal{N}_S$. The sort sets are constructed according to the type construction in which the name has been used. Entity and relationship types

are associated with predicate variables. The logical language uses type expansion for representation of relationship types. We can use key-based expansion or full expansion. Full expansion uses all components of the component type. Key-based expansion uses only (primary) key components. If all names are different in S then we can use lowercase strings for variable names. If this is not the case then we use a dot notation similar to the record notation in programming languages.

The HERM predicate logic is inductively constructed in the same way as the predicate logic. Instead of simple variables we use structured variables. This language enables us to specify restrictions on the scheme. *Queries* can be expressed in a similar way.

We can also specify behavior of a database over lifetime. A database is modified by an action or more general by a transaction. Basic actions are queries or conditional manipulation operations. Manipulation operations such as *insert, delete, update* are defined in the HERM algebra. Database behavior can be specified on the basis of states. Given a HERM scheme $S = \{R_1, ...R_m\}$. A *state* is the set of classes $\{R_1^C, ...R_m^C\}$ with $R_i^C \in \mathcal{R}(R_i)$, $1 \leq i \leq m$ which satisfies certain restrictions Σ.

The structuring of the extended ER model allows to deduct a number of properties. As an example we consider the axiomatisation of constraints generalizing those discussed in [Tha91]. We observe first that implication in the hierarchical predicate logic is reflexive, monotone, compact and closed. Let us consider classes of BV-constraints in HERM which form a cylindric algebra [Tsa89]. The order of constraints by $\Phi_{\models}$ possibly can be based on the order of of premises and conclusions. In this case the constraint set forms a pair algebra.

Observation 7
Cylindric classes are pair algebras.

Examples of cylindric classes are the class of functional dependencies, the classes of Hungarian functional dependencies [Tha91], the class of inclusion dependencies and the class of multivalued dependencies. Further, the n-class of all $\geq n$-functional dependencies $X \rightarrow Y$ which left side contains at least n components and the class of rigid $\leq n$-inclusion dependencies $T_1[X] \subseteq T_2[X]$ which component list contain at most n components form a cylindric constraint set. Usually, union does not preserve cylindric sets.

Observation 8
Cylindric constraint classes are axiomatised by reflexivity axioms, augmentation and transition rules.

If an axiomatisation leads to reflexivity, augmentation and transitivity then union and decomposition rules can be deducted by the other rules. Transitivity may have to consider the specific influence of premises, e.g., transitivity for full multivalued dependencies is based on the root reduction rule [Tha91].

Based on this axiomatisation we may introduce a general vertical decomposition form:
Given a schema structuring $\mathcal{S} = (\mathcal{ER}, \Sigma_{\mathcal{S}})$. A *vertical decomposition* of $\mathcal{S}$ is given a a mapping τ from $\mathcal{S}$ to $\mathcal{S}'$ which is defined by projection functions. The decomposition is *lossless* if a query q on $\mathcal{S}'$ can be defined such that for each db on $\mathcal{S}$ the equality

$q(\tau(db)) = db$ is valid.
Let further Σ' the set of those constraints from $\Phi_{\models}(\Sigma)$ which are entirely defined on the structures in $\mathcal{S}'$. A decomposition based on projection is called *$\mathcal{C}$-constraint preserving* if $\Sigma \subseteq \Phi_{\models}(\Sigma')$.

Classical example of vertical decompositions are decompositions of relations to relations in the third normal form.

We may now introduce a general class of $\mathcal{C}$-decomposition algorithms:

Construct basic elements which are undecomposable.
Derive maximal elements by backward propagation of augmentation.
Reduce redundancy in the constraint set by backward propagation of transitivity.
Derive a left-right graph by associating conclusions of a constraint with the premise of another constraint.
Combine all minimal left sides of constraints which are not bound by another constraint to a group.
Derive projections based on all groups in the graph.

The first step of the decomposition algorithm is only introduced for convenience. This algorithm is a generalization of the classical synthesis algorithm.
Observation 9
The $\mathcal{C}$-decomposition algorithm leads to $\mathcal{C}$-constraint preserving decomposition if the class $\mathcal{C}$ is cylindric.

2.4 Maturity and Capability of Object-Oriented/Object-Relational Models

Object-oriented database models have been developed in order to overcome the impedance mismatch between languages for specification of structural aspects and languages for the specification of behavioral aspects. So far, no standard approach is known to object-orientation. *Objects* are handled in databases systems and specified on the basis of database models. They can own an *object identifier*, are structurally characterized by *values* and *references* to other objects and can posses their own *methods*, i.e.

$$o = (i, \{v\}, \{ref\}, \{meth\})$$

The value characterization is bound to a *structure* of the type T which is already defined. Characterizing properties of objects are described by *attributes* which form the structure of the object. Objects also have a *specific semantics* and a *general semantics*. The properties describe the *behavior* of objects. Objects which have the same structure, the same general semantics and the same operators are collected in *classes*. The structure, the operations and the semantics of a class are represented by *types* $T = (S, O, \Sigma)$. In this case, the modelling of objects includes the association of objects with classes C and their corresponding value type T and reference type R. Therefore, after classification the structure of objects is represented by

$$o = (i, \{(C, T, v)\}, \{(C, R, ref)\}, \{(T, meth)\}).$$

The recognized design methodologies vary in the scale of information modelled in the types. If objects in the classes can be distinguished by their values, then the identifiers can be omitted and we use *value-oriented modelling*. If this is not the case, we

use an *object-oriented approach*. In the object-oriented approach, different approaches can be distinguished. If all objects are identifiable by their value types or by references to identifiable objects, then the database is called *value-representable*. In this case, the database can also be modelled by the *value-oriented* approach, and a mapping from the value-representable scheme to a value-oriented scheme can be generated. If the database is not value-representable, then we have to use object identifiers. In this case either the identifier handling should be made public or else the databases cannot be updated and maintained. Therefore, value-representable databases are of special interest. Thus, we can distinguish database models as displayed in Figure 1.

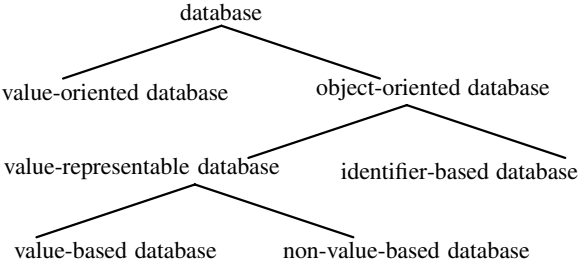


Fig. 1. Classification of Database Models

It has been shown in [BT99, Sch94] that the concept of the object identifier can only be treated on the basis of higher-order epistemic and intuitionistic logics. Furthermore, identification by identifiers is different from identification by queries, equational logics and other identification methods. For this reason, the concept of the object identifier is far more complex than wanted and cannot be consistently and equivalently treated in database systems. Furthermore, methods can be generically derived from types only in the case if all objects are value-representable. Value-representable cyclic type systems require topos semantics[ST99] what is usually too complex to be handled in database systems. It can be shown that value-representable, non-cyclic type systems can be represented by value-oriented models.

2.5 XML - Couleur De Rose and Pitfalls

XML document specification and playout is based on a number of standards and co-standards:

XML documents are based on trees of elementary documents which are tagged by a begin and end delimiter.
Document schema specification is either based on
- DTD specification which supports an entity-relationship modelling,
- RDF schema reuse which allows to include other specifications, or
- Schema specification which allows object-oriented modelling.

XLink, XPointer, XPath and XNamespace support a flexible parsing and playout enabling documents to be extended by foreign sources which might be accessible.
XSL supports filtering, union and join of documents.
XML query languages add query facilities to XML document suites.
XML applications are supported by a large variety of application standards such as BizTalk and LOM.

This layering has a number of advantages and a number of disadvantages. The main advantage of this approach is that almost any object set can be represented by an XML document suite. XML documents may be well-formed. In this case they are semi-structured and may be represented by $\mathcal{A}$-trees which are defined by induction as follows
- each $\sigma \in \mathcal{A}$ is a tree, and
- if $\sigma \in \Sigma$ and $t_1, ..., t_n$ are trees then $\sigma(t_1, ..., t_n)$ is a tree.

The set $Dom(t) \subseteq \mathcal{N}^*$ of all nodes of a tree $t = \sigma(t_1, ..., t_n)$ is given by:

$$Dom(t) = \{\epsilon\} \cup \{ui | i \in \{1, ..., n\}, u \in Dom(t_i)\}$$

where ϵ is the root, ui is the ith child of u, and
$lab^t(u)$ is the label of u in t.

The disadvantages of XML steam from the generality of the approach. For instance, parsing of XML document sets must be supported by machines which are not less complex than Turing machines, i.e., tree automata
$M = (Q, \Sigma, \delta, (I_\sigma)_{\sigma \in \Sigma}, F), \quad F \subseteq Q, \delta : Q \times \Sigma \to 2^{Q^*}$.
A run $\lambda : Dom(t) \to Q$ specifies for each leave node $v : \epsilon \in \delta(\lambda(u1), lab^t(u))$ and for each node v with p children : $\lambda(u1)\lambda(u2)...\lambda(up) \in \delta(\lambda(u), lab^t(u))$.
The run accepts the tree t if $\lambda(\epsilon) \in F$.

XML document suites have, however, a number of other properties: they are partial and based on list semantics. Their implication is neither compact not monotone nor closed. Therefore, the axiomatisation of XML constraints is more difficult compared with other database models. For instance, already the definition of keys and functional dependencies becomes a nightmare. The treatment of cardinality constraints is more difficult than for ER models. For instance, the definitions of [AFL02, BDF$^+$01] are incomplete since they do not consider the list-to-set axiom.

XML documents provide a universal structuring mechanism. [Kle07] has developed a modelling approach that limits the pitfalls of XML specification.

Finite implication of path constraints is co-r.e. complete and implication is r.e. complete for semi-structured models. The implication problem for key constraints is harder than in the relational and ER case. It involves implication on regular path expressions which is known to be PSPACE-hard. The satisfiability problem for key constraints and referential inclusion constraints becomes undecidable in the presence of DTD's. For this reason, simpler language must be used for specification of constraints in XML.

3 Overuse and Misuse of the Notion of Semantics

Nowadays semantics is viewed in a large variety of ways. Some of them are [HR04]: semantic web, semantics as a metamodel, semantics as context, semantics as behaviour,

semantics as being executable, semantics as the meaning of individual constructs, semantics as mathematical notation, and semantics as mappings. All these views have their own pitfalls and do not convey with the notion of semantics. They are mainly syntactic types that 'cry out for' [Bjø06] a semantic explanation.

3.1 Semantification of Semantic Web

The "Semantic Web" is mainly based on syntax and partially uses micro-semantics of wordings. Semantics is used in the sense of rudimentary lexical semantics. Rudimentary lexical semantics must be enhanced by explicit definitions of symbols or words used. These definitions can be combined with the name spaces that provide a source for the lexical units used in a web document. The semantification project of the group headed by J. Pokorny at Charles University Prague aims in enhancing the ontology based annotation in XML documents by a semantic repository, by user profiles and by portfolio management.

Web documents should be enhanced by context [KSTZ03] or meta-data similar to the OpenCyc project. Lexical units may be characterised by *time(absolut, type), place(absolut, type), culture, sophistification/security, topic/usage, granularity, modality/disposition/epistemology, argument preferences, justification,* and *lets* [Len02].

The vocabulary of name spaces or of ontologies is not just a collection of words scattered at random throughout web documents. It is at least partially structured, and at various levels. There are various modes and ways of structuring, e.g., by branching, taxonomic or meronymic hierarchies or by linear bipole, monopolar or sequenced structures.

Ontologies are often considered to be the silver bullet for web integration. They are sometimes considered to be an explicit specification of conceptualisation or to be a shard understanding of some domain of interest that can be communicated across people and computers. We should however distinguish a variety of ontologies such as generic, semiotic, intention/extension, language, UoD, representational, context and abstraction ontologies.

3.2 Separation of Semantics and Behaviour

A common misconception in computer science is confusion of semantics and behaviour. Behaviour is an aspect in systems description. It is also based on syntax, semantics and pragmatics. We use dynamic algebraic structures of a certain signature for the description of behaviour.

Typical aspects concentrate either on models describing structural properties or evolution or the collaboration among actors or the distribution or architecture of the system. Figure 2 surveys aspects we might consider.

Aspects describe different separate concerns.

Structuring of a database application is concerned with representing the database structure and the corresponding static integrity constraints.

Behaviour of a database application is specified on the basis of processes and dynamic integrity constraints.

Distribution of information system components is specified through explicit specification of services and exchange frames.

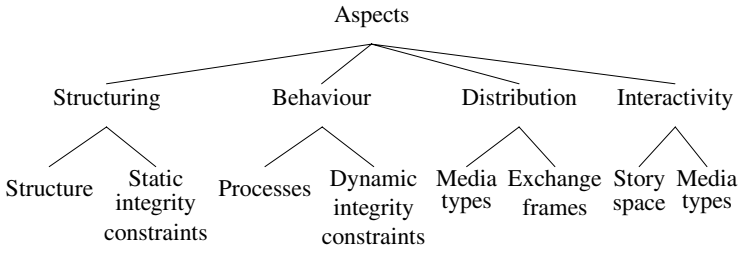


Fig. 2. Description, prescription and design aspects of the WIS

Interactivity is provided by the system on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

This understanding has led to the co-design approach to modelling by specification *structuring*, *behaviour*, *distribution*, and *interactivity*. These four aspects of modelling have both syntactic and semantic elements.

3.3 Semantics Illustration through Diagrams

The Unified Modelling Language UML consists of more than 100 different types of diagrams. These diagrams provide some illustration for syntactic constructs. They are partially semantically founded. Their meaning is often only given by an abstract description of the intended meaning or by an ad-hoc polymorphism of constructs.

Diagrams make use of pictured elements. These elements are used in diagrams in very different ways and meanings. As an example one might compare the meaning of the arrow or the edge in different UML diagrams. The meaning of these elements is implicitly defined by the kind of diagram. The same problem can be observed for assignment of cardinality constraints to edges in ER diagrams despite the differences between participation and look-through semantics. Consistency of elements used in different diagrams is still an open issue.

Diagrams provide a simple way for visual representation of structures, functions, interaction or distribution. Visualization is, however, not the silver bullet as often marketed. It may mislead, e.g. by misleading comparisons or by overuse or wrong use of coloring schemes that vary in cultural environments. Representation of complex structures, e.g., in medicine cannot be entirely based on visual structures. Reasoning on representations, e.g., in UML diagrams is not yet supported. Diagrams may misguide as well as values without context.

The most important drawback of diagrams is their spatial restriction due to the requirement that diagrams must be surveyable. This requirements causes a separation of content into an ensemble of diagrams. The consistency of diagrams must thus be explicitly handled. So far there is no theory that allows to handle consistency in diagram ensembles such as UML diagram ensembles.

4 Contributions of SDKB to Data and Knowledge Base Semantics

This volume continues the 'Semantics in Databases' workshops. The first workshop has been published in Lecture Notes in Computer Science 1358 and the second in Lecture Notes in Computer Science 2582. The third workshop was collocated with EDBT 2008. We are thankful to the EDBT organisers in Nantes. At the workshop six papers selected from 19 papers that have been submitted. Two contributions that reflect the topic of semantics in databases and in knowledge bases have also been invited to this workshop. This volumes additionally contains two invited papers.

Already with the second workshop we realised that the approaches to semantics in database research and in knowledge base research are similar and may enhance each other. The introduction [BKST03] surveys different approaches to semantics in database research and discusses a program to research on database semantics. Most of these research questions as well as the problems in [Tha87, Tha00a, LT98] are still open.

The papers in this volume reflect a variety of approaches to semantics in data and knowledge bases:

A. Altuna develops a model-theoretic semantics for the formalisation of context in knowledge bases. The approach accommodates reasoning across the different perspectives that may coexist on a particular theory or situation. This approach allows an explicit distinction between the context and the interpretation for formal semantics as discussed above.

D. W. Archer and L. M. L. Delcambre propose an explicit capture model for data integration decisions. This approach allows to handle data integration in a similar way as users do while learning the content of a document ensemble. Users evolve their understanding by restructuring, copying, pasting, abstracting and extending them by their context. The paper shows how this integration of lexical, logical, prototype and dynamic semantics supports entity resolution (record linkage, de-duplication, data identification) and de-resolution.

R. Caballero and Y. García-Ruiz, and F. Sáenz-Pérez develop a theoretical framework for debugging Datalog programs based on the ideas of declarative debugging. Whenever a user detects an unexpected answer for some query the debugger allows to track the cause for missing and wrong answers. The debugging mechanism is based on computation graphs and debugging circuits.

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati and M. Ruzzi introduce a complete system for data integration that uses an ontology and description logic as the main vehicle. Data integration supports a uniform access to a set of heterogeneous data sources and does not require that the user knows implementation details. Lexical semantics can efficiently (LogSpace) be combined in an with a conceptual architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema.

F. A. Ferrarotti and J. M. Turull Torres invesitgate the connection between the concept of relational complexity and the restricted second-order logic SO^{ω}. This logic allows to characterize the relational polynomial-time hierarchy. The existential fragment of SO^{ω} captures relational NP. Prenex fragments of SO^{ω} exacly corresponds to the levels of the relational polynomial-time hierarchy.

S. Hartmann, H. Köhler, S. Link, T. Trinh, and J. Wang survey the variety of proposals for the notion of keys in XML documents. Typically XML documents are defined by application of list type constructor to basic types. Therefore, semantics is different from the case of typical database modelling languages. Additionally, XML documents have optional components. The notion of the key must thus be redefined. There are several notions for keys in XML documents. They are partially incomparable, differ in their properties and expressive power.

Stephen J. Hegner develops a general technique for establishing that the translation of a view update defined by constant complement is independent of the choice of complement. Views provide partial information. The treatment of semantics by views has been one of the open problems in [BKST03]. It is shown that the decomposition mapping for the pair of views is required not only to be bijective on states but also on the sentences which define the information content of the component views.

Gabriele Kern-Isberner, Matthias Thimm and Marc Finthammer develop a theory of truth values beyond classical two-valued semantics based on degrees of plausibility that express how strongly a formula is supported. Conditionals serve as a basic means to encode plausible, intensional relationships in the form of general rules. An algebraic theory of conditional structures is presented which provides the formal machinery to make use of conditionals for inference and belief revision in different frameworks. The approach is used for the development of a theory of qualitative knowledge discovery.

N. Mohajerin and N. Shiri use query rewriting techniques for the development of a top-down approach for answering queries using only answers to views. On the basis a graph-based model for conjunctive queries and views, the algorithm proposed in the paper efficiently generates maximally contained rewritings which are in general less expensive to evaluate, compared to the bottom-up algorithms, without requiring post-processing. Existing solutions that follow a bottom-up approach require a post-processing phase.

Héctor Pérez-Urbina, Boris Motik and Ian Horrocks develop a resolution-based algorithm for rewriting conjunctive queries over TBoxes in Description Logic DL-Lite$^+$. The algorithm produces an optimal rewriting when the input ontology is expressed in the language DL-Lite. The complexity is NLogSpace w.r.t. data complexity. Combining this result with the lower bound it is concluded that query answering in DL-Lite+ is NLogSpace-complete.

References

[AFL02] Arenas, M., Fan, W., Libkin, L.: On veriyfying consistency of XML specifications. In: Proc. ACM PODS, pp. 259–270. ACM, New York (2002)

[Bak95] De Bakker, J.: Control flow semantics. MIT Press, Cambridge (1995)

[BDF$^+$01] Buneman, P., Davidson, S., Fan, W., Hara, C.S., Tan, W.C.: Keys for XML. In: Proc. WWW 10, pp. 201–210. ACM Press, New York (2001)

[Bjø06] Bjørner, D.: Software Engineering 2: Specification of systems and languages. Springer, Berlin (2006)

[BKST03] Bertossi, L., Katona, G.O.H., Schewe, K.-D., Thalheim, B.: Semantics in databases. In: Bertossi, L., Katona, G.O.H., Schewe, K.-D., Thalheim, B. (eds.) Semantics in Databases 2001. LNCS, vol. 2582, pp. 1–6. Springer, Heidelberg (2003)

[BS03] Börger, E., Stärk, R.: Abstract state machines - A method for high-level system design and analysis. Springer, Berlin (2003)

[BST06] Bienemann, A., Schewe, K.-D., Thalheim, B.: Towards a theory of genericity based on government and binding. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 311–324. Springer, Heidelberg (2006)

[BT99] Beeri, C., Thalheim, B.: Identification as a primitive of database models. In: Proc. FoMLaDO 1998, pp. 19–36. Kluwer, London (1999)

[Cry87] Crystal, D.: The Cambridge encyclopedia of language. Cambridge University Press, Cambridge (1987)

[DHT+08] Duzi, M., Heimburger, A., Tokuda, T., Vojtas, P., Yoshida, N.: Multi-agent knowledge modelling. Panel summary, EJC 2008 (2008)

[DM00] Duží, M., Materna, P.: Constructions (2000), http://til.phil.muni.cz/text/constructions_duzi_materna.pdf

[Gun92] Gunther, C.: Semantics of programming languages. MIT Press, Cambridge (1992)

[Hau01] Hausser, R.: Foundations of computational linguistics: Human-computer communication in natural language, 2nd edn. Springer, Berlin (2001)

[HR04] Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer, 64–72 (October 2004)

[Kle07] Klettke, M.: Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen. Advanced PhD (Habilitation Thesis), Rostock University, Faculty for Computer Science and Electronics (2007)

[KSTZ03] Kaschek, R., Schewe, K.-D., Thalheim, B., Zhang, L.: Integrating context in conceptual modelling for web information systems, web services, e-business, and the semantic web. In: Bussler, C.J., Fensel, D., Orlowska, M.E., Yang, J. (eds.) WES 2003. LNCS, vol. 3095, pp. 77–88. Springer, Heidelberg (2004)

[Len02] Lenat, D.: The dimensions of the context space (2002), http://www.cyc.com/context-space.pdf

[Lin03] Link, S.: Consistency enforcement in databases. In: Bertossi, L., Katona, G.O.H., Schewe, K.-D., Thalheim, B. (eds.) Semantics in Databases 2001. LNCS, vol. 2582, pp. 139–159. Springer, Heidelberg (2003)

[LT98] Libkin, L., Thalheim, B.: Foreword. In: Thalheim, B. (ed.) Semantics in Databases 1995. LNCS, vol. 1358, pp. V–IX. Springer, Heidelberg (1998)

[Mos92] Mosses, P.D.: Action semantics. Cambridge University Press, Cambridge (1992)

[PBGG89] Paredaens, J., De Bra, P., Gyssens, M., Van Gucht, D.: The structure of the relational database model. Springer, Berlin (1989)

[Rey99] Reynolds, J.: The semantics of programming languages. Cambridge University Press, Cambridge (1999)

[Sch71] Schwabhäuser, W.: Modelltheorie I. B.I.-Wissenschaftsverlag, Mannheim (1971)

[Sch72] Schwabhäuser, W.: Modelltheorie II. B.I.-Wissenschaftsverlag, Mannheim (1972)

[Sch94] Schewe, K.-D.: The specification of data-intensive application systems. Advanced PhD (Habilitation Thesis), Brandenburg University of Technology at Cottbus, Faculty of Mathematics, Natural Sciences and Computer Science (1994)

[ST99] Schewe, K.-D., Thalheim, B.: A generalization of Dijkstra's calculus to typed program specifications. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 463–474. Springer, Heidelberg (1999)

[ST08] Schewe, K.-D., Thalheim, B.: Life cases: A kernel element for web information systems engineering. In: WEBIST 2007. LNBIP, vol. 8, pp. 139–156. Springer, Berlin (2008)

[Ste73] Stegmüller, W.: Theorie und Erfahrung, Zweiter Halbband: Theorienstrukturen und Theoriendynamik. In: Probleme und Resultate der Wissenschaftstheorie und Analytischen Philosophie, vol. II. Springer, Heidelberg (1973)

[Ten97] Tennent, R.: The semantics of programming languages. Prentice-Hall, Englewood Cliffs (1997)

[Tha87] Thalheim, B.: Open problems in relational database theory. Bull. EATCS 32, 336–337 (1987)

[Tha91] Thalheim, B.: Dependencies in relational databases. Teubner, Leipzig (1991)

[Tha00a] Thalheim, B.: Entity-relationship modeling – Foundations of database technology. Springer, Berlin (2000)

[Tha00b] Thalheim, B.: The person, organization, product, production, ordering, delivery, invoice, accounting, budgeting and human resources pattern in database design. Technical Report Preprint I-07-2000, Brandenburg University of Technology at Cottbus, Institute of Computer Science (2000)

[Tsa89] Sh, M.: Tsalenko. Modeling of semantics for databases. Nauka, Moscov (1989) (in Russian)

[Wan87] Wanner, P. (ed.): The Cambridge Encyclopedia of Language. Cambridge University Press, Cambridge (1987)

[Web91] Websters ninth new collegiate dictionary (1991)

[Who80] Whorf, B.L.: Lost generation theories of mind, language, and religion. Popular Culture Association, University Microfilms International, Ann Arbor, Mich. (1980)

[Wit58] Wittgenstein, L.: Philosophical Investigations. Basil Blackwell, Oxford (1958)

Data Integration through *DL-Lite*$_{\mathcal{A}}$ Ontologies

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Maurizio Lenzerini[2], Antonella Poggi[2], Riccardo Rosati[2], and Marco Ruzzi[2]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano
calvanese@inf.unibz.it
[2] Dip. di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

Abstract. The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. One of the outcomes of the research work carried out on data integration in the last years is a clear conceptual architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema. In this paper, we present a comprehensive approach to, and a complete system for, ontology-based data integration. In this system, the global schema is expressed in terms of a TBox of the tractable Description Logics *DL-Lite*$_{\mathcal{A}}$, the sources are relations, and the mapping language allows for expressing GAV sound mappings between the sources and the global schema. The mapping language has specific mechanisms for addressing the so-called impedance mismatch problem, arising from the fact that, while the data sources store values, the instances of concepts in the ontology are objects. By virtue of the careful design of the various languages used in our system, answering unions of conjunctive queries can be done through a very efficient technique (LOGSPACE with respect to data complexity) which reduces this task to standard SQL query evaluation. We also show that even very slight extensions of the expressive abilities of our system lead beyond this complexity bound.

1 Introduction

The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. The problem of designing effective data integration solutions has been addressed by several research and development projects in the last years. Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32,26]. One of the outcomes of this research work is a clear conceptual architecture for (mediator-based[1]) data integration. According to this architecture [26], the main components of a data integration system are the global schema, the sources, and the mapping. Thus, a data integration system is seen as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

[1] Other architectures, e.g. [4], are outside the scope of this paper.

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 26–47, 2008.

- $\mathcal{G}$ is the *global schema*, providing both a conceptual representation of the application domain, and a reconciled, integrated, and virtual view of the underlying sources.
- $\mathcal{S}$ is the *source schema*, i.e., the schema of the sources where real data are stored.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of assertions establishing the connection between the elements of the global schema and those of the source schema. Two basic approaches have been proposed in the literature. The first approach, called *global-as-view* (or simply GAV) [11, 18, 20, 31], focuses on the elements of the global schema, and associates to each of them a view (query) over the sources. On the contrary, in the second approach, called *local-as-view* (or simply LAV) [10, 15, 23], the focus is on the sources, in the sense that a view (query) over the global schema is associated to each of them.

We use the term "data integration management system" to denote a software tool supporting the conceptual architecture described above. Among the various services to be provided by a data integration management system, we concentrate on query answering: Queries are posed in terms of the global schema, and are to be answered by suitably reasoning on the global schema, and exploiting the mappings to access data at the sources.

Data integration is still one of the major challenges in Information Technology. One of the reasons is that large amounts of heterogeneous data are nowadays available within an organization, but these data have been often collected and stored by different applications and systems. Therefore, the need of accessing data by means of flexible and unified mechanisms is becoming more and more important. On the other hand, current commercial data integration tools have several drawbacks. In particular, none of them realizes the goal of describing the global schema independently from the sources. In particular, these tools do not allow for specifying integrity constraints in the global schema, and this implies that the global schema is a sort of data structure for accommodating a reconciled view of the source data, rather than a faithful description of the application domain. It follows that current state-of-the-art data integration tools do not support the conceptual architecture mentioned above.

In this paper, we present a comprehensive approach to, and a complete management system for ontology-based data integration. The system, called MASTRO-I, is based on the following principles:

- The system fully adheres to the conceptual architecture developed by the scientific community.
- The global schema is specified in terms of an ontology, specifically in terms of a TBox expressed in a tractable Description Logics, namely *DL-Lite*$_{\mathcal{A}}$. So, our approach conforms to the view that the global schema of a data integration system can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system.
- The source schema is the schema of a relational database. In fact, such a schema may result from the federation of a set of heterogeneous, possibly non-relational, data sources. This can be realized by means of a data federation tool, which presents, without materializing them, physical data sources to MASTRO-I as they

were a single relational database, obtained by simply transforming each source into a set of virtual relational views and taking their union.

– The mapping language allows for expressing GAV *sound* mappings between the sources and the global schema. A GAV sound mapping specifies that the extension of a source view provides a subset of the tuples satisfying the corresponding element of the global schema.
  Moreover, the mapping language has specific mechanisms for addressing the so-called *impedance mismatch* problem. This problem arises from the fact that, while the data sources store values, the instances of concepts in the ontology (global schema) are objects, each one denoted by an identifier (e.g., a constant in logic), not to be confused with any data item.

MASTRO-I is based on the system QUONTO [1], a reasoner for *DL-Lite*$_{\mathcal{A}}$, and is coupled with a data federation tool that is in charge of federating physical data sources[2].

We point out that our proposal is not the first one advocating the use of ontologies in data integration (see, for example, [2,12]). However, to the best of our knowledge, MASTRO-I is the first data integration management system addressing simultaneously the following aspects:

– providing a solution to the impedance mismatch problem;
– answering unions of conjunctive queries posed to the global schema according to a method which is sound and complete with respect to the semantics of the ontology;
– careful design of the various languages used in the system, resulting in a very efficient technique (LOGSPACE with respect to data complexity), which reduces query answering to standard SQL query evaluation over the sources.

In order to demonstrate feasibility and efficiency of the MASTRO-I approach to data integration, we also describe in this paper an experimentation carried out over a real-world application scenario. More precisely, we discuss some experiments in which we make use of an ontology benchmark modeling the domain of universities to specify the global schema of the integration system, and connect it, via MASTRO-I mappings, to real data stored at different information systems owned by SAPIENZA University of Rome.

Although in the present work we make use of GAV mappings, the presence of constraints expressed in a rich ontology language in the global schema, makes query answering in our setting more similar to what is carried out in LAV data integration systems rather than in GAV systems. Indeed, while in general GAV systems have been realized as (simple) hierarchies of wrappers and mediators, query answering in LAV can be considered a form of reasoning in the presence of incomplete information, and thus significantly more complex. Early systems based on this approach, like Information Manifold (IM) [28,29], or INFOMASTER [16,19], have implemented algorithms [28] for rewriting queries using views, where the views are the ones specified through the

[2] The current implementation of MASTRO-I makes use of Websphere Federation Server, the IBM tool for data federation (http://www-306.ibm.com/software/data/integration/federation_server/). However, MASTRO-I can be coupled with any data federation tool based on SQL.

(conjunctive) queries in the mappings. The relationship between LAV and GAV data integration systems is explored in [5], where it is indeed shown that a LAV system can be converted into a GAV one by introducing suitable inclusion dependencies in the global schema. If no functionality assertions are present in the global schema, such inclusion dependencies can then be dealt with in a way similar to what is done here for concept and role inclusions in *DL-Lite*$_{\mathcal{A}}$. We show in this paper, however, that this is no longer possible in the presence of functionality assertions.

Indeed, one might wonder whether the expressive power of the data integration framework underlying MASTRO-I can be improved. We answer this question by showing that even very slight extensions of the expressive abilities of MASTRO-I in modeling the three components of a data integration system lead beyond the LOGSPACE complexity bound.

We finally point out that MASTRO-I addresses the problem of data integration, and not the one of schema or ontology integration. In other words, MASTRO-I is not concerned with the task of building the ontology representing the global schema starting from the source schema, or from other ontologies. This task, which is strongly related to other important problems, such as database schema integration [3], and ontology alignment, matching, merging, or integration (see, e.g., [17]), are outside the scope of MASTRO-I.

The paper is organized as follows. In Section 2, we describe in detail the various components of the data integration framework adopted in MASTRO-I. In Section 3, we provide a description of the semantics of a data integration system managed by MASTRO-I. In Section 4, we illustrate the basic characteristics of the query answering algorithm. In Section 5, we present our experiments. Finally, in Section 6 we study possible extensions to the MASTRO-I framework, and in Section 7 we conclude the paper.

2 The Data Integration Framework

In this section we instantiate the conceptual architecture for data integration systems introduced in Section 1, by describing the form of the global schema, the source schema, and the mapping for data integration systems managed by MASTRO-I.

2.1 The Global Schema

Global schemas managed by MASTRO-I are given in terms of TBoxes expressed in *DL-Lite*$_{\mathcal{A}}$ [30], a DL of the *DL-Lite* family. Besides the use of concepts and roles, denoting sets of objects and binary relations between objects, respectively, *DL-Lite*$_{\mathcal{A}}$ allows one to use value-domains, a.k.a. concrete domains, denoting unbounded sets of (data) values, and concept attributes, denoting binary relations between objects and values[3]. In particular, the value-domains that we consider here are those corresponding to unbounded (i.e., value-domains with an unbounded size) RDF data types, such as integers, real, strings, etc.

[3] In fact, all results presented in [30] and exploited in the present paper can be extended to include role attributes, user-defined domains, as well as inclusion assertions over such domains (see also [7]).

To describe *DL-Lite*$_{\mathcal{A}}$, we first introduce the DL *DL-Lite*$_{\mathcal{FR}}$, which combines the main features of two DLs presented in [8], called *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, respectively. We use the following notation:

- A denotes an *atomic concept*, B a *basic concept*, C a *general concept*, and $\top_C$ the *universal concept*;
- E denotes a *basic value-domain*, i.e., the range of an attribute, $T_1, \ldots, T_n$ denote the n pairwise disjoint *unbounded RDF data types* used in our logic, and F denotes a *general value-domain*, which can be either an unbounded RDF data type T_i or the *universal value-domain* $\top_D$;
- P denotes an *atomic role*, Q a *basic role*, and R a *general role*;
- U denotes an *atomic attribute*, and V a *general attribute*.

Given an attribute U, we call *domain* of U, denoted by $\delta(U)$, the set of objects that U relates to values, and we call *range* of U, denoted by $\rho(U)$, the set of values related to objects by U.

We are now ready to define *DL-Lite*$_{\mathcal{FR}}$ expressions as follows.

- Basic and general *concept expressions*:

$$B \ ::= \ A \ \mid \ \exists Q \ \mid \ \delta(U) \qquad\qquad C \ ::= \ B \ \mid \ \neg B$$

- Basic and general *value-domain expressions*:

$$E \ ::= \ \rho(U) \qquad\qquad F \ ::= \ \top_D \ \mid \ T_1 \ \mid \ \cdots \ \mid \ T_n$$

- General *attribute expressions*:

$$V \ ::= \ U \ \mid \ \neg U$$

- Basic and general *role expressions*:

$$Q \ ::= \ P \ \mid \ P^- \qquad\qquad R \ ::= \ Q \ \mid \ \neg Q$$

A *DL-Lite*$_{\mathcal{FR}}$ *TBox* allows one to represent intensional knowledge by means of assertions of the following forms:

- *Inclusion assertions*:

$B \sqsubseteq C$	*concept inclusion assertion*
$Q \sqsubseteq R$	*role inclusion assertion*
$E \sqsubseteq F$	*value-domain inclusion assertion*
$U \sqsubseteq V$	*attribute inclusion assertion*

  A concept inclusion assertion expresses that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions.
- *Functionality assertions* on atomic attributes or basic roles:

$$(\mathsf{funct}\ I) \qquad \textit{functionality assertion}$$

  where I denotes either an atomic attribute or a basic role.

DL-Lite$_{\mathcal{A}}$ TBoxes are *DL-Lite*$_{\mathcal{FR}}$ TBoxes with suitable limitations in the combination of *DL-Lite*$_{\mathcal{FR}}$ TBox assertions. To describe such limitations we first introduce some preliminary notions.

An atomic attribute U (resp., an atomic role P) is called a *functional property in a TBox* $\mathcal{T}$, if $\mathcal{T}$ contains a functionality assertion $(\mathsf{funct}\ U)$ (resp., either $(\mathsf{funct}\ P)$ or $(\mathsf{funct}\ P^-)$). Then, an atomic attribute is called *primitive in* $\mathcal{T}$, if it does not appear positively in the right-hand side of an attribute inclusion assertion of $\mathcal{T}$, i.e., an atomic attribute U_1 is primitive in $\mathcal{T}$ if there does not exist an atomic attribute U_2 such that $U_2 \sqsubseteq U_1$ is asserted in $\mathcal{T}$. Similarly, an atomic role is called *primitive in* $\mathcal{T}$ if it or its inverse do not appear positively in the right-hand side of a role inclusion assertion of $\mathcal{T}$, i.e., an atomic role P is primitive in $\mathcal{T}$ if there does not exist a basic role Q such that neither $Q \sqsubseteq P$ nor $Q \sqsubseteq P^-$ is asserted in $\mathcal{T}$.

Then, *a DL-Lite*$_{\mathcal{A}}$ *TBox is a DL-Lite*$_{\mathcal{FR}}$ *TBox* $\mathcal{T}$ *satisfying the condition that every functional property in* $\mathcal{T}$ *is primitive in* $\mathcal{T}$.

Roughly speaking, in our logic, *functional properties cannot be specialized*, i.e., they cannot be used positively in the right-hand side of role/attribute inclusion assertions. As shown in [30], reasoning over a *DL-Lite*$_{\mathcal{A}}$ knowledge base (constituted by a TBox and an ABox, which specifies the instances of concept and roles) is tractable. More precisely, TBox reasoning is in PTIME and query answering is in LOGSPACE w.r.t. data complexity, i.e., the complexity measured in the size of the ABox only (whereas query answering for *DL-Lite*$_{\mathcal{FR}}$ is PTIME-hard [8]). Thus, *DL-Lite*$_{\mathcal{A}}$ presents the same computational behavior of all DLs of the *DL-Lite* family, and therefore is particularly suited for integration of large amounts of data.

2.2 The Source Schema

The source schema in MASTRO-I is assumed to be a flat relational database schema, representing the schemas of all the data sources. Actually, this is not a limitation of the system, since the source schema coupled with MASTRO-I can be the schema managed by a data federation tool. So, the data federation tool is in charge of interacting with data sources, presenting them to MASTRO-I as a single relational database schema, obtained by wrapping physical sources, possibly heterogeneous, and not necessarily in relational format. Furthermore, the data federation tool is in charge of answering queries formulated over the source schema, by suitably transforming and asking them to the right sources, finally combining the single results into the overall answer. In other words, the data federation tool makes MASTRO-I independent from the physical nature of the sources, by providing a logical representation of them (physical independence), whereas MASTRO-I is in turn in charge of making all logical aspects transparent to the user, by maintaining the conceptual global schema separate from the logical federated schema, and connecting them via suitable mappings (logical independence).

2.3 The Mapping

The mapping in MASTRO-I establishes the relationship between the source schema and the global schema, thus specifying how data stored at the sources are linked to the instances of the concepts and the roles in the global schema. To this aim, the mapping

specification takes suitably into account the impedance mismatch problem, i.e., the mismatch between the way in which data is (and can be) represented in a data source, and the way in which the corresponding information is rendered through the global schema.

The MASTRO-I mapping assertions keep data value constants separate from object identifiers, and construct identifiers as (logic) terms over data values. More precisely, object identifiers in MASTRO-I are *terms* of the form $f(d_1, \ldots, d_n)$, called object terms, where f is a function symbol of arity $n > 0$, and $d_1, \ldots, d_n$ are data values stored at the sources. Note that this idea traces back to the work done in deductive object-oriented databases [22].

We detail below the above ideas. The mapping in MASTRO-I is specified through a set of mapping assertions, each of the form

$$\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$$

where

- $\Phi(\boldsymbol{v})$, called the *body* of the mapping, is a first-order logic (FOL) query of arity $n > 0$, with distinguished variables $\boldsymbol{v}$, over the source schema $\mathcal{S}$ (we will write such query in the SQL syntax), and
- $P(\boldsymbol{w})$, called the *head*, is an atom where S can be an atomic concept, an atomic role, or an atomic attribute occurring in the global schema $\mathcal{G}$, and $\boldsymbol{w}$ is a sequence of terms.

We define three different types of mapping assertions:

1. *Concept mapping assertions*, in which the head is a unary atom of the form $A(f(\boldsymbol{v}))$, where A is an atomic concept and f is a function symbol of arity n;
2. *Role mapping assertions*, in which the head is a binary atom of the form $P(f_1(\boldsymbol{v}'), f_2(\boldsymbol{v}''))$, where P is an atomic role, f_1 and f_2 are function symbols of arity $n_1, n_2 > 0$, and $\boldsymbol{v}'$ and $\boldsymbol{v}''$ are sequences of variables appearing in $\boldsymbol{v}$;
3. *Attribute mapping assertions*, in which the head is a binary atom of the form $U(f(\boldsymbol{v}'), v'' : T_i)$, where U is an atomic attribute, f is a function symbol of arity $n' > 0$, $\boldsymbol{v}'$ is a sequence of variables appearing in $\boldsymbol{v}$, v'' is a variable appearing in $\boldsymbol{v}$, and T_i is an RDF data type.

In words, such mapping assertions are used to map source relations (and the tuples they store), to concepts, roles, and attributes of the ontology (and the objects and the values that constitute their instances), respectively. Note that an attribute mapping also specifies the type of values retrieved by the source database.

Example 1. Consider the following mapping assertions:

M_1 : `SELECT S_CODE FROM STUDENTS WHERE DOB <= '1990/01/01'` $\rightsquigarrow$ *Student*(**st**(`S_CODE`))

M_2 : `SELECT S_CODE, S_NAME FROM STUDENTS WHERE DOB <= '1990/01/01'` $\rightsquigarrow$ *name*(**st**(`S_CODE`), `S_NAME : xsd:string`)

M_3 : `SELECT S_CODE, C_CODE FROM COURSES` $\rightsquigarrow$ *takesCourse*(**st**(`S_CODE`), **co**(`C_CODE`))

Such assertions relate a source schema containing the relations STUDENTS and COURSES to a global schema containing the atomic concept *Student*, the atomic attribute *name*, and the atomic role *takesCourse*. M_1 is a concept mapping assertion that selects from the table STUDENTS the code (variable S_CODE) of students whose date of birth (variable DOB) is after December 31, 1989, and for each such code builds, by means of the function symbol **st**, an object term representing an instance of the concept *Student*. M_2 is an attribute mapping assertion that, besides the codes, selects also the names (variable S_NAME) of the students born after December 31, 1989, and for each selected tuple builds a pair constituted by a term of the form **st**(S_CODE), which is as in assertion M_1, and a constant representing the name of the student. To such a constant M_2 assigns the data type `xsd:string`. Finally, M_3 is a role mapping assertion relating the relation COURSES to the global atomic role *takesCourse*. More precisely, from each pair constituted by the code of a student (variable S_CODE) and the code of a course she takes (variable C_CODE), M_3 builds a pair of object terms (**st**(S_CODE),**co**(C_CODE)), where **co** is a function symbol used to build object terms representing courses taken by students.

We point out that, in fact, the body of each mapping assertion is never really evaluated in order to extract values from the sources to build instances of the global schema, but rather it is used to unfold queries posed over the global schema, and thus rewriting them into queries posed over the source schema (cf. Section 4). Also, we notice that the mapping designer has to specify a correct *DL-Lite*$_\mathcal{A}$ data type for the values extracted from the source, in order to guarantee coherency of the system. This aspect is detailed in the next section.

3 Semantics

We now illustrate the semantics of a data integration system managed by MASTRO-I.

Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system. The general idea is to start with a database D for the source schema $\mathcal{S}$, i.e., an extension of the data sources, called the *source database* for $\mathcal{J}$. The source database D has to be coherent with the typing assertions that implicitly appears in the mapping $\mathcal{M}$. More precisely, this means that, for every attribute mapping $\Phi(\boldsymbol{v}) \rightsquigarrow U(f(\boldsymbol{v'}), v'' : T_i)$, the values for v'' extracted from D are of type T_i. In the rest of this paper, we always assume that the source database is coherent with the mapping.

Given a source database D, we define the semantics of $\mathcal{J}$ as the set of interpretations for $\mathcal{G}$ that both satisfy the TBox assertions of $\mathcal{G}$, and satisfy the mapping assertions in $\mathcal{M}$ with respect to D. This informal definition makes use of different notions that we detail below.

- First, the notion of interpretation for $\mathcal{G}$ is the usual one in DL. An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ for $\mathcal{G}$ consists of an interpretation domain $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$. $\Delta^\mathcal{I}$ is the disjoint union of the domain of objects $\Delta^\mathcal{I}_O$, and the domain of values $\Delta^\mathcal{I}_V$, while the interpretation function $\cdot^\mathcal{I}$ assigns the standard formal meaning to all expressions and assertions of the logic *DL-Lite*$_\mathcal{A}$ (see [7]). The only aspect which is not standard here is the need of dealing with objects denoted by terms (see

previous section). To this end, we now introduce two disjoint alphabets, called Γ_O and Γ_V, respectively. Symbols in Γ_O are called object terms, and are used to denote objects, while symbols in Γ_V, called value constants, are used to denote data values. More precisely, Γ_O is built starting from Γ_V and a set Λ of function symbols of any arity (possibly 0), as follows: If $f \in \Lambda$, the arity of f is n, and $d_1, \ldots, d_n \in \Gamma_V$, then $f(d_1, \ldots, d_n)$ is a term in Γ_O, called *object term*. In other words, object terms are either functional terms of arity 0, called object constants, or terms constituted by a function symbol applied to data value constants. We are ready to state how the interpretation function $\cdot^{\mathcal{I}}$ treats Γ_V and Γ_O: $\cdot^{\mathcal{I}}$ assigns a different value in $\Delta_V^{\mathcal{I}}$ to each symbol in Γ_V, and a different element of $\Delta_O^{\mathcal{I}}$ to every object term (not only object constant) in Γ_O. In other words, *DL-Lite*$_{\mathcal{A}}$ enforces the unique name assumption on both value constants and object terms.

– To the aim of describing the semantics of mapping assertions with respect to a database D for the source schema $\mathcal{S}$, we first assume that all data values stored in the database D belong to Γ_V[4]. Then, if q is a query over the source schema $\mathcal{S}$, we denote by $ans(q, D)$ the set of tuples obtained by evaluating the query q over the database D (if q has no distinguished variables, then $ans(q, D)$ is a boolean). Finally, we introduce the notion of ground instance of a formula. Let γ be a FOL formula with free variables $\boldsymbol{x} = (x_1, \ldots, x_n)$, and let $\boldsymbol{s} = (s_1, \ldots, s_n)$ be a tuple of elements in $\Gamma_V \cup \Gamma_O$. A ground instance $\gamma[\boldsymbol{x}/\boldsymbol{s}]$ of γ is obtained from γ by substituting every occurrence of x_i with s_i.
We are now ready to specify the semantics of mapping assertions. We say that an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* the mapping assertion $\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$ with respect to D, if for every ground instance

$$\Phi[\boldsymbol{v}/\boldsymbol{s}] \rightsquigarrow S[\boldsymbol{v}/\boldsymbol{s}]$$

of $\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$, we have that $ans(\Phi[\boldsymbol{v}/\boldsymbol{s}], D) = true$ implies $S[\boldsymbol{v}/\boldsymbol{s}]^{\mathcal{I}} = true$, where, for a ground atom $S(\boldsymbol{t})$, with $\boldsymbol{t} = (t_1, \ldots, t_n)$ a tuple of terms, we have that $S(\boldsymbol{t})^{\mathcal{I}} = true$ if and only if $(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \in p^{\mathcal{I}}$. Note that the above definition formalizes the notion of sound mapping, as it treats each mapping assertion as an implication.
– With the above notion in place, we define the semantics of $\mathcal{J}$ with respect to D as follows:

$$sem_D(\mathcal{J}) = \{ \text{ interpretation } \mathcal{I} \mid \mathcal{I} \text{ satisfies all assertions in } \mathcal{G} \text{ and } \mathcal{M} \text{ wrt } D \}$$

We say that $\mathcal{J}$ is satisfiable withe respect the D if $sem_D(\mathcal{J}) \neq \emptyset$.

Among the various reasoning services that can be considered over a data integration system, in this paper we are interested in the problem of answering unions of conjunctive queries (UCQs) posed to the global schema. The semantics of query answering is given in terms of certain answers to the query, defined as follows. Given a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a database D for $\mathcal{S}$, the set of *certain answers* to the query $q(\boldsymbol{x})$ over $\mathcal{G}$ with respect to D (denoted by $cert(q, \mathcal{J}, D)$) is the set of all tuples $\boldsymbol{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that $q[\boldsymbol{x}/\boldsymbol{t}]$ is true in every $\mathcal{I} \in sem_D(\mathcal{J})$.

[4] We could also introduce suitable conversion functions in order to translate values stored at the sources into value constants in Γ_V, but we do not deal with this issue here.

4 Query Answering

In this section, we sketch our query answering technique (more details can be found in [30]). Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a database D for $\mathcal{S}$, and assume that $\mathcal{J}$ is satisfiable with respect to D, i.e., $sem_D(\mathcal{J}) \neq \emptyset$.

We start with the following observation. Suppose we evaluate (over D) the queries in the left-hand sides of the mapping assertions, and we materialize accordingly the corresponding assertions in the right-hand sides. This would lead to a set of ground assertions, that can be considered as a *DL-Lite* ABox[5], denoted by $\mathcal{A}^{\mathcal{M},D}$. It can be shown that query answering over $\mathcal{J}$ and D can be reduced to query answering over the *DL-Lite*$_\mathcal{A}$ knowledge base constituted by the TBox $\mathcal{G}$ and the ABox $\mathcal{A}^{\mathcal{M},D}$. However, due to the materializion of $\mathcal{A}^{\mathcal{M},D}$, the query answering algorithm resulting from this approach would be polynomial in the size of D. On the contrary, our idea is to avoid any ABox materialization, but rather answer Q by reformulating it into a new query that can be afterwards evaluated directly over the database D. The resulting query answering algorithm is constituted by four steps, which are called *rewriting*, *filtering*, *unfolding* and *evaluation*, and are described in the following.

4.1 Rewriting

Given a UCQ Q over a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a source database D for $\mathcal{J}$, the rewriting step computes a new UCQ Q' over $\mathcal{J}$, where the assertions of $\mathcal{G}$ are compiled in. In computing the rewriting, only inclusion assertions of the form $B_1 \sqsubseteq B_2$, $Q_1 \sqsubseteq Q_2$, and $U_1 \sqsubseteq U_2$ are taken into account, where B_i, Q_i, and U_i, with $i \in \{1, 2\}$, are a basic concept, a basic role and an atomic attribute, respectively. Intuitively, the query Q is rewritten according to the knowledge specified in $\mathcal{G}$ that is relevant for answering Q, in such a way that the rewritten query Q' is such that $cert(Q', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D) = cert(Q, \mathcal{J}, D)$, i.e., the rewriting allows to get rid of $\mathcal{G}$.

Example 2. Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{G}$ is *DL-Lite*$_\mathcal{A}$ TBox comprising the concept inclusion assertion $\mathit{Student} \sqsubseteq \exists \mathit{takesCourse}$, and consider the query $Q(x) :- \mathit{takesCourse}(x, y)$ (written in Datalog syntax), which is asking for individuals that take (at least) a course. The output of the rewriting step is the following UCQ Q' (written in Datalog syntax):

$$Q'(x) :- \mathit{takesCourse}(x, y).$$
$$Q'(x) :- \mathit{Student}(x).$$

Since the global schema says that each student takes at least a course, the query Q' asks both for individuals that take a course and for individuals that are students. Then, we can answer Q' over $\mathcal{J}$ by disregarding the inclusion assertions in $\mathcal{G}$, and we get the same result we got by answering Q over $\mathcal{J}$. In other words, whatever is the underlying source database D, we have that $cert(Q', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D) = cert(Q, \mathcal{J}, D)$.

[5] In DL jargon, an ABox is a set of membership assertions, i.e., assertions stating which are the instances of the concepts and the roles defined in the corresponding TBox.

We refer the reader to [30,9] for a formal description of the query rewriting algorithm used by MASTRO-I and for a proof of its soundness and completeness. We only notice here that the rewriting procedure does not depend on the source database D, runs in polynomial time in the size of $\mathcal{G}$, and returns a query Q' whose size is at most exponential in the size of Q.

4.2 Filtering

Let Q' be the UCQ produced by the rewriting step above. In the filtering step we take care of a particular problem that the disjuncts, i.e., conjunctive queries, in Q' might have. Specifically, a conjunctive query cq is called *ill-typed* if it has at least one join variable x appearing in two incompatible positions in cq, i.e., such that the TBox $\mathcal{G}$ of our data integration system logically implies that x is both of type T_i, and of type T_j, with $T_i \neq T_j$ (remember that in *DL-Lite*$_\mathcal{A}$ data types are pairwise disjoint). The purpose of the filtering step is to remove from the UCQ Q' all the ill-typed conjunctive queries. Intuitively, such a step is needed because the query Q' has to be unfolded and then evaluated over the source database D (cf. the next two steps of the MASTRO-I query answering algorithm described below). These last two steps, performed for an ill-typed conjunctive query might produce incorrect results. Indeed, the set of certain answers over a satisfiable data integration system for an ill-typed conjunctive query cq is always empty (cf. Section 3). However, the SQL query that results after the unfolding step (see below) is sent to the underlying data federation tool that uses its SQL engine for evaluating it over the source database, and it might happen that the data value conversions carried out by the SQL engine make the evaluation of such an SQL query non-empty, thus incorrectly producing a non-empty set of certain answers for cq. Obviously, this might produce an incorrect result in all those cases in which cq occurs in the UCQ Q'. The filtering step, by simply dropping all ill-typed queries from Q', solves this problem and produces a new UCQ Q'' over $\mathcal{J}$.

Example 3. Consider the boolean conjunctive query

$$Q :- \mathit{id}(x, z), \mathit{age}(x, z).$$

where *id* and *age* are two atomic attributes, asking if there exists a value constant which is both the *id* and the *age* of a certain individual (e.g., a student). Consider now a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{M}, \mathcal{S} \rangle$ where $\mathcal{G}$ contains the assertions

$$\begin{aligned} \rho(\mathit{id}) &\sqsubseteq \texttt{xsd:string} \\ \rho(\mathit{age}) &\sqsubseteq \texttt{xsd:integer} \end{aligned}$$

specifying that the range of *id* is `xsd:string`, and the range of *age* is `xsd:integer`. Obviously, the query Q above is ill-typed, since `xsd:string` and `xsd:integer` are two disjoint data types.

Let us now answer Q over $\mathcal{J}$, and assume that the output of the rewriting step is Q itself (i.e., the global schema does not contain assertions that cause the rewriting of Q). If we skipped now the filtering step, the query above would be handed to the unfolder (see below), which would transform it into an SQL query (according to $\mathcal{M}$).

Then, its evaluation over some source database D might return some tuples, due to some data value conversions that make the join specified in Q succeed. In other words, $cert(Q, \mathcal{J}, D)$ might be non-empty and the query might incorrectly be considered true. The filtering step, instead, simply drops the ill-typed queries from the UCQ to be sent to the unfolder, which in this example is therefore an empty query. As a consequence, $cert(Q, \mathcal{J}, D)$ is correctly empty.

4.3 Unfolding

Given the UCQ Q'' over $\mathcal{J}$ computed by the filtering step, the unfolding step computes, by using logic programming technology, an SQL query Q''' over the source schema $\mathcal{S}$, that possibly returns object terms. It can be shown [30] that Q''' is such that $ans(Q''', D) = cert(Q'', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D)$, i.e., unfolding allows us to get rid of $\mathcal{M}$. Moreover, the unfolding procedure does not depend on D, runs in polynomial time in the size of $\mathcal{M}$, and returns a query whose size is polynomial in the size of Q''.

Example 4. Let us now continue Example 2, and assume that $\mathcal{M}$ is constituted by the mapping assertions described in Example 1. It is easy to see that Q' does not contain ill-typed queries, and therefore the outut of the filtering step is Q' itself. Also, it is easy to see that Q' has to be unfolded by means of mapping assertions M_1 (used to unfold $Q(x) :- \mathit{takesCourse}(x, y)$) and M_3 (used to unfold $Q(x) :- \mathit{Student}(x)$). The SQL query that results from this unfolding is the following:

```
SELECT CONCAT('st(',S_CODE,')') FROM COURSES
UNION
SELECT CONCAT('st(',S_CODE,')') FROM STUDENTS
WHERE DOB <= '1990/01/01'
```

Notice that in the above query we have made use of the SQL function `CONCAT`[6]. Such a function allows us to concatenate strings to construct object terms of the form **st**(`S_CODE`). By virtue of this mechanism, the evaluation of the above query over the source database returns indeed a set of object terms representing individuals (in fact, students), coherently to what the original query Q asks for.

4.4 Evaluation

The evaluation step consists in simply delegating the evaluation of the SQL query Q''', produced by the unfolding step, to the data federation tool managing the data sources. Formally, such a tool returns the set $ans(Q''', D)$, i.e., the set of tuples obtained from the evaluation of Q''' over D.

4.5 Correctness of Query Answering

It can be shown that the query answering procedure described above correctly computes the certain answers to UCQs. Based on the computational properties of such an algorithm, we can then characterize the complexity of our query answering method.

[6] For simplicity we assume that the underlying data federation tool allows for using `CONCAT` with an arbitrary number of arguments.

Theorem 1. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system, and D a source database for $\mathcal{J}$. Answering a UCQ over $\mathcal{J}$ with respect to D can be reduced to the evaluation of an SQL query over D, and is* LOGSPACE *in the size of D.*

Finally, we remark that, as we said at the beginning of this section, we have assumed that the data integration system $\mathcal{J}$ is consistent with respect to the database D, i.e., $sem_D(\mathcal{J})$ is non-empty. Notably, it can be shown that all the machinery we have devised for query answering can also be used for checking consistency of $\mathcal{J}$ with respect to D. Therefore, checking consistency can also be reduced to sending appropriate SQL queries to the source database [30].

5 Experimentation

In this section, we comment on the results of an experimentation that we have carried out on a real-world data integration scenario. The main aim of the experimentation is to test on a case of real and practical interest the MASTRO-I architecture for data integration, which reflects the fundamental principle of maintaining separate the physical level of the data sources, which remain autonomous and independent, from the conceptual representation of the domain of discourse, whose design reflects only the ambit of interest and is in principle independent from the specific data at the sources. To this aim, we have considered a set of information sources used by different administrative offices of SAPIENZA University of Rome, and we have used the so-called Lehigh University Benchmark (LUBM)[7] to specify the global schema of our integration system. LUBM consists of an ontology for modeling universities, and it is a de facto standard for benchmarking ontology reasoners. Usually, extensional data for the LUBM intensional level are synthesized in an automatic way, possibly using benchmark generators available for LUBM. Here, instead, by virtue of the mapping mechanism provided by MASTRO-I, we are able to connect our specific data sources to the LUBM ontology, which has been of course designed independently from such data.

5.1 Scenario

As data sources, we consider three legacy relational databases, containing the overall number of 25 relations, each storing from a few tuples up to 50,000 tuples. We make use of IBM Websphere Federation Server to federate the above data sources, in such a way that MASTRO-I can see such sources as a single relational schema (source schema). As already said, to model the global schema we make use of the LUBM ontology. The ontology contains concepts for persons, students, professors, publications, courses, etc., as well as appropriate relationships for such a universe of discourse. The ontology is specified in OWL-DL[8], and it currently defines 43 classes and 32 properties (including 25 object properties, i.e., roles, and 7 datatype properties, i.e., attributes). Note that, in order to use the LUBM ontology in MASTRO-I, we rephrased it in *DL-Lite*$_{\mathcal{A}}$, essentially

[7] http://swat.cse.lehigh.edu/projects/lubm/
[8] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

capturing all its constructs[9], and also enriched it with further TBox assertions modeling peculiar aspects of the domain not present in the original ontology. For example, we also considered the role $hasExam$, to model the courses for which a student has passed the exam.

Due to space limits, we provide below only a fragment of the *DL-Lite*$_{\mathcal{A}}$ ontology used in our experiments.

$$\begin{array}{rcl@{\qquad}rcl} \mathit{Student} & \sqsubseteq & \exists \mathit{takesCourse} & \exists \mathit{takesCourse}^- & \sqsubseteq & \mathit{Course} \\ \mathit{Course} & \sqsubseteq & \exists \mathit{teacherOf}^- & \exists \mathit{teacherOf} & \sqsubseteq & \mathit{Faculty} \\ \mathit{Faculty} & \sqsubseteq & \exists \mathit{worksFor} & \exists \mathit{worksFor}^- & \sqsubseteq & \mathit{University} \\ \mathit{University} & \sqsubseteq & \exists \mathit{hasAlumnus} & \exists \mathit{hasAlumnus}^- & \sqsubseteq & \mathit{Student} \end{array}$$

In words, the assertions in the first column, from top to bottom, respectively say that each student must take a course (i.e., must be connected by the role *takesCourse* to a certain individual), each course is necessarily taught by some individual, each faculty participates to the role *worksFor*, each university has at least one alumnus. Assertions in the second column, from top to bottom, respectively say that individuals in the inverse of the role *takesCourse* (resp. the role *teacherOf*, the inverse of the role *worksForUniv*, and the inverse of the role *hasAlumnus*) must be courses (resp. faculties, universities, students).

5.2 Testing Query Answering

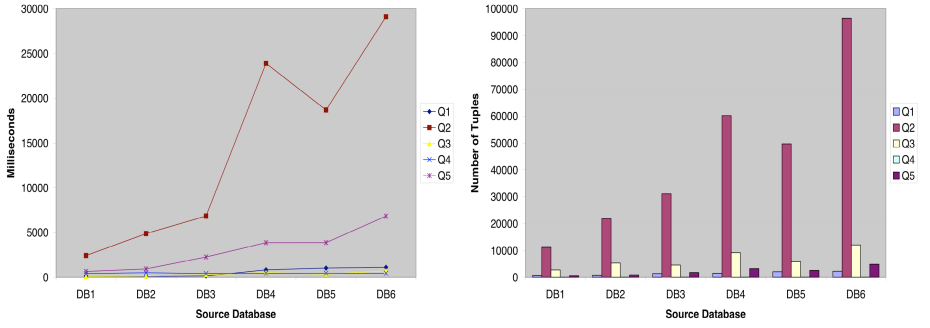
To show scalability of query answering in MASTRO-I, we tuned our system in such a way that data stored at the sources resulted into six different source databases of growing size. Table 1 briefly says how data coming from the information systems of SAPIENZA University of Rome have been filtered for our experiments.

Table 1. Source databases used for tests

Name	DB size (number of tuples)	Data description
DB_1	118075	from 1993 to 1995 (restricted to students living in Rome)
DB_2	165049	from 1993 to 1995
DB_3	202305	from 1993 to 1997 (restricted to students living in Rome)
DB_4	280578	from 1993 to 1997
DB_5	328256	from 1993 to 1999 (restricted to students living in Rome)
DB_6	482043	from 1993 to 1999

We then considered five significant queries over the global schema, and measured the behavior of MASTRO-I in terms of both the size of the resulting answer sets (i.e., the number of tuples in the answer to each query), and the overall time that MASTRO-I took to produce these answer sets. Below we describe each test query (given in DATALOG notation).

[9] We recall that, since *DL-Lite*$_{\mathcal{A}}$ is less expressive than OWL-DL, an OWL-DL ontology cannot be in general exactly specified in *DL-Lite*$_{\mathcal{A}}$.



(a) (b)

Fig. 1. Query answering: (a) Execution time (b) Number of tuples in the answer

– *Query 1*: It asks for all persons living in Rome

$$Q_1(x) :- \mathit{Person}(x), \mathit{address}(x, \texttt{'ROMA'}).$$

– *Query 2*: It asks for the names of all students that take a course, together with the name of such a course:

$$Q_2(z, w) :- \mathit{Student}(x), \mathit{name}(x, z), \mathit{takesCourse}(x, y), \mathit{name}(y, w).$$

– *Query 3*: It asks for all persons that passed at least an exam:

$$Q_3(x) :- \mathit{Person}(x), \mathit{hasExam}(x, y).$$

– *Query 4*: It asks for the names of all persons whose address is the same as the address of the organization for which their advisor works:

$$Q_4(z) :- \mathit{Person}(y), \mathit{name}(y, z), \mathit{address}(y, w), \mathit{advisor}(y, x), \mathit{worksFor}(x, v), \mathit{address}(v, w).$$

– *Query 5*: It asks for all students that take a course, together with the address of the organization for which the course teacher works:

$$Q_5(x, c) :- \mathit{Student}(x), \mathit{takesCourse}(x, y), \mathit{teacherOf}(z, y), \mathit{worksFor}(z, w), \mathit{address}(w, c).$$

The results of our experiments are given in Figure 1, which shows the performance (execution time) for answering each query w.r.t. the growth of the size of the source database (Figure 1(a)), and the number of tuples returned by each query for each source database (Figure 1(b)).

All experiments have been carried out on an Intel Pentium IV Dual Core machine, with 3 GHz processor clock frequency, equipped with 1 Gb of RAM, under the operating system Windows XP professional.

5.3 Discussion

For each query, the execution time comprises the time needed for rewriting and filtering the input query (both rewriting and filtering are executed by the system QUONTO, which is at the core of MASTRO-I) unfolding the resulting query, and evaluating the unfolded query over the underlying database (both unfolding and evaluation are delegated to IBM Websphere Federation Server). We point out that the time needed for query rewriting and query unfolding is negligible w.r.t. the overall execution time, and that the major time consuming process is the evaluation of the rewritten and unfolded query over the source database. This depends both on the number of disjuncts occurring in the rewritten query (which is a union of conjunctive queries), and the number of source relations mapped to concepts, roles, and attributes occurring as predicates of the query atoms. As an example, we provide below the rewriting of Query 2 (expressed in Datalog notation), for which we measured the worst performance in terms of execution times (n_0 below denotes a fresh existentially quantified variable introduced by the rewriting process).

$$
\begin{array}{lcl}
Q_2(z, w) & :- & \mathit{name}(y, w), \mathit{examRating}(x, y, n_0), \mathit{name}(x, z). \\
Q_2(z, z) & :- & \mathit{takesGraduateCourse}(x, x), \mathit{name}(x, z). \\
Q_2(z, w) & :- & \mathit{name}(y, w), \mathit{takesGraduateCourse}(x, y), \mathit{name}(x, z). \\
Q_2(z, w) & :- & \mathit{name}(y, w), \mathit{hasExam}(x, y), \mathit{name}(x, z). \\
Q_2(z, z) & :- & \mathit{examRating}(x, x, n_0), \mathit{name}(x, z). \\
Q_2(z, z) & :- & \mathit{takesCourse}(x, x), \mathit{name}(x, z). \\
Q_2(z, z) & :- & \mathit{hasExam}(x, x), \mathit{name}(x, z). \\
Q_2(z, w) & :- & \mathit{name}(y, w), \mathit{takesCourse}(x, y), \mathit{name}(x, z).
\end{array}
$$

We notice that MASTRO-I shows good scalability w.r.t. the growth of the size of the ABox, and that execution time is always limited, even for answering queries that are rewritten into unions of conjunctive queries with several disjuncts.

6 Extending the Data Integration Framework

In this section we study whether the data integration setting presented above can be extended while keeping the same complexity of query answering. In particular, we investigate possible extensions for all the three components $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ of the system.

6.1 Extensions to *DL-Lite*$_\mathcal{A}$

With regard to the logic used to express the global schema $\mathcal{G}$, the results in [8] already imply that it is not possible to go beyond *DL-Lite*$_\mathcal{A}$ (at least by means of the usual DL constructs) and at the same time keep the data complexity of query answering within LOGSPACE. Here we consider the possibility of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of a data integration system, both two distinct value constants, and two distinct object terms denote two different domain elements. Unfortunately, this leads query answering out of LOGSPACE, as shown by the following theorem.

Theorem 2. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system extended by removing the UNA, and D a database for $\mathcal{S}$. Answering a UCQ (in fact, a query constituted by a single atom) over $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Proof. We prove the result already for a data integration system in which the global schema is expressed in *DL-Lite*$_{\mathcal{F}}$ [8], a sub-language of *DL-Lite*$_{\mathcal{A}}$, and in which the mapping assertions have the simplest possible form, i.e., they map a single source relation to a single concept or role of the global schema[10]. The proof is based on a reduction from reachability in directed graphs, which is NLOGSPACE-hard.

Let $G = \langle V, E \rangle$ be a directed graph, where V is the set of vertexes and E the set of directed edges. Reachability is the problem of deciding, given two vertexes $s, t \in V$ whether there is an oriented path formed by edges in E in the graph that, starting from s allows to reach t. We consider the graph represented through first-child and next-sibling functional relations F, N, S (cf. Figure 2).

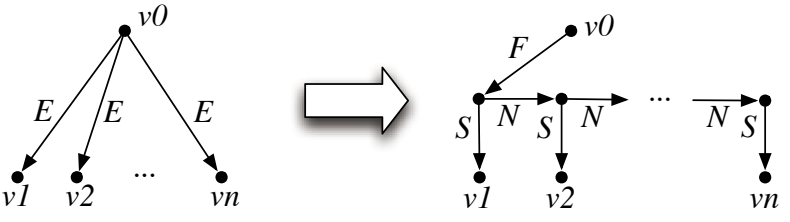


Fig. 2. Representation of a graph through the functional relations F, N, S

We define the data integration system $\mathcal{J}_{una} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ as follows:

- The alphabet of $\mathcal{G}$ consists of the atomic concept A_g and the atomic roles F_g, N_g, S_g, and P_g. $\mathcal{G}$ consists only of the functionality assertions $\{(\mathsf{funct}\ \mathcal{R}_g) \mid \mathcal{R}_g \in \{F_g, N_g, S_g, P_g\}\}$.
- $\mathcal{S}$ contains the binary relational tables F_s, N_s, S_s, and P_s, with columns c_1 and c_2, and the unary relational table A_s, with column c.
- The mapping $\mathcal{M}$ maps each table $\mathcal{R}_s$ to the corresponding role or concept $\mathcal{R}_g$, i.e.,

$$\begin{array}{lcll} \texttt{SELECT}\ c_1, c_2\ \texttt{FROM}\ \mathcal{R}_s & \rightsquigarrow & \mathcal{R}_g(\mathbf{id}(c_1), \mathbf{id}(c_2)), & \text{for } \mathcal{R} \in \{F, N, S, P\} \\ \texttt{SELECT}\ c\ \texttt{FROM}\ A_s & \rightsquigarrow & A_g(\mathbf{id}(c)) & \end{array}$$

  Notice that we are using a single function symbol **id** (that we intend to represent the identity function).

Then, from the graph G and the two vertexes s, t, we define the source database D_G as follows:

$$\begin{array}{rcl} D_G & = & \{\mathcal{R}_s(a, b), \mathcal{R}_s(a', b') \mid (a, b) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F, N, S\}\} \cup \\ & & \{P_s(init, s), P_s(init, s')\} \cup \{A_s(t)\} \end{array}$$

[10] The mapping assertions actually play no role in the proof, and the hardness result holds already for a plain *DL-Lite*$_{\mathcal{F}}$ knowledge base constituted by a TBox and an ABox.

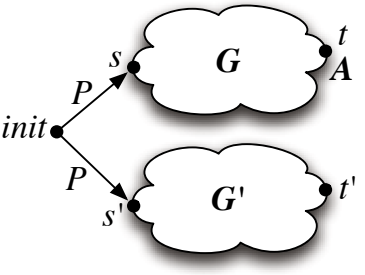


Fig. 3. Structure of the database used in the proof of Theorem 2

In other words, we encode in D_G two copies of (the representation of) the graph G. In addition, we include in D_G the facts $A_s(t)$, $P_s(init, s)$, and $P_s(init, s')$, where $init$ is a database constant that does not correspond to any vertex of (the representation of) G (cf. Figure 3).

It is now possible to prove that t is reachable from s in G iff $\mathbf{id}(t') \in cert(Q, \mathcal{J}_{una}, D_G)$, where $Q(x) :- A_g(x)$ is the query returning the instances of A_g. Indeed, it is easy to verify that the latter holds if and only if $\mathbf{id}(t)$ and $\mathbf{id}(t')$ are the same object in every interpretation in $sem_{D_G}(\mathcal{J}_{una})$, i.e., the equality $\mathbf{id}(t) = \mathbf{id}(t')$ is entailed by $\mathcal{J}_{una}$. This is the case if and only if $\mathbf{id}(t)$ and $\mathbf{id}(t')$ are forced to be equal by the functionality of the roles P_g, F_g, N_g, and S_g. Given the structure of the database D_G, such an equality is enforced if and only if t is reachable from s in G. □

Notice that a simple variation of the above proof can be used to show that query answering, and in particular instance checking already, in *DL-Lite*$_{\mathcal{F}}$ without the unique name assumption is NLOGSPACE-hard with respect to data complexity.

6.2 Different Source Schemas

Although MASTRO-I is currently only able to deal with relational sources, managed by a relational data federation tool, it is not hard to see that all the results presented in this paper apply also if we consider federation tools that provide a representation of the data at the sources according to a different data model (e.g., XML). Obviously, depending on the specific data model adopted by the data federation tool, we have to resort to a suitable query language for expressing the source queries appearing in the mapping assertions. To adhere to our framework, the only constraint on this language is that it is able to extract tuples of values from the sources, a constraint that is trivially satisfied by virtually all query languages used in practice.

6.3 Extensions to the Mapping Language

As for the language used to express the mapping $\mathcal{M}$, we investigate the extension of the mapping language to allow for GLAV assertions, i.e., assertions that relate conjunctive queries over the sources to conjunctive queries over the global schema. Such assertions are therefore an extension of both GAV and LAV mappings. Unfortunately, even with

LAV mappings only, instance checking and query answering are no more in LOGSPACE wrt data complexity, as the following theorem shows.

Theorem 3. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system extended with LAV mapping assertions, and D a database for $\mathcal{S}$. Answering a UCQ (in fact, a query constituted by a single atom) over $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Proof. The proof is again by a reduction from reachability in directed graphs. Let $G = \langle V, E \rangle$ be a directed graph, where V is the set of vertexes and E the set of directed edges. Again, we consider the graph represented through first-child and next-sibling functional relations F, N, S (cf. Figure 2).

We define the data integration system $\mathcal{J}_{lav} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ as follows:

- The alphabet of $\mathcal{G}$ consists of the atomic concept A_g and the atomic roles F_g, N_g, S_g, P_g, and $copy_g$. $\mathcal{G}$ consists only of the functionality assertions $\{(\mathsf{funct}\ \mathcal{R}_g) \mid \mathcal{R}_g \in \{F_g, N_g, S_g, P_g, copy_g\}\}$.
- $\mathcal{S}$ contains the binary relational tables F_s, N_s, and S_s, with columns c_1 and c_2, and the unary relational table A_s, with column c.
- The LAV mapping $\mathcal{M}$ is defined as follows (cf. Figure 4)[11]:

$$\begin{array}{rcl} A_s(x) & \rightsquigarrow & q_1(x) :- A_g(x), copy_g(x, x'), P_g(z, x), P_g(z, x') \\ \mathcal{R}_s(x, y) & \rightsquigarrow & q_2(x, y) :- \mathcal{R}_g(x, y), copy_g(x, x'), copy_g(y, y'), \mathcal{R}_g(x', y'), \\ & & \qquad \text{for } \mathcal{R} \in \{F, N, S\} \end{array}$$

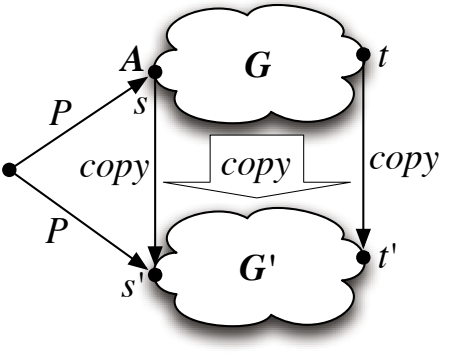


Fig. 4. Interpretation generated by the LAV mapping used in the proof of Theorem 3

Then, from the graph G and the two vertexes s, t, we define the source database D_G as follows:

$$D_G = \{\mathcal{R}_s(a, b) \mid (a, b) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F, N, S\}\} \cup \{A_s(s)\}$$

[11] For simplicity, we do not include function symbols in the mapping since, as in the proof of Theorem 2, they would play no role in the reduction.

Intuitively, D_G is simply constituted by the binary relations F_s, N_s, and S_s, used to represent the graph G, and a unary relation A_s containing s.

Now consider the query $Q(x) :- copy_g(x, x)\}$. Then, it is possible to show that t is reachable from s in G iff $t \in cert(Q, \mathcal{J}_{lav}, D_G)$. □

7 Conclusions

We close the paper by briefly mentioning some aspects that have been considered important for the problem of (ontology-based) data integration, but that have not been addressed in the present paper, and are left for future work on the system MASTRO-I.

A first important point is handling inconsistencies in the data, possibly using a declarative, rather than an adhoc procedural approach. An interesting proposal is the one of the INFOMIX system [27] for the integration of heterogeneous data sources (e.g., relational, XML, HTML) accessed through a relational global schema with powerful forms of integrity constraints. The query answering technique proposed in such a system is based on query rewriting in Datalog enriched with negation and disjunction, under stable model semantics [6,21]. A first study on how to adapt the semantics for consistent query answering to *DL-Lite* ontologies can be found in [25].

A further aspect is that of instance level integration and mappings, which deals with the situation where individual instances, rather than ontology elements, in different sources need to be mapped to each other (cf., e.g, [24]).

Finally, one notable direction for further work is making MASTRO-I a "write-also" data integration tool. Indeed, while the present version of MASTRO-I provides support for answering queries posed to the data integration system, it is of interest to also deal with updates expressed on the global schema (e.g., according to the approach described in [13,14]. The most challenging issue to be addressed in this context is to design mechanisms for correctly reformulating an update expressed over the ontology into a series of insert and delete operations on the data sources.

Acknowledgments. This research has been partially supported by the FET project TONES (Thinking ONtologiES), funded by the EU under contract number FP6-7603, by project HYPER, funded by IBM through a Shared University Research (SUR) Award grant, and by the MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

1. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 1670–1671 (2005)
2. Amann, B., Beeri, C., Fundulaki, I., Scholl, M.: Ontology-based integration of XML web resources. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 117–131. Springer, Heidelberg (2002)
3. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys 18(4), 323–364 (1986)

4. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In: Proc. of the 5th Int. Workshop on the Web and Databases (WebDB 2002) (2002)
5. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503. Springer, Heidelberg (2002)
6. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic DL-Lite$_A$. In: Proc. of the 2nd Workshop on OWL: Experiences and Directions (OWLED 2006) (2006)
8. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning 39(3), 385–429 (2007)
10. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. of Cooperative Information Systems 10(3), 237–271 (2001)
11. Carey, M.J., Haas, L.M., Schwarz, P.M., Arya, M., Cody, W.F., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., Thomas, J., Williams, J.H., Wimmers, E.L.: Towards heterogeneous multimedia information systems: The Garlic approach. In: Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM 1995), pp. 124–131. IEEE Computer Society Press, Los Alamitos (1995)
12. Catarci, T., Lenzerini, M.: Representing and using interschema knowledge in cooperative information systems. J. of Intelligent and Cooperative Information Systems 2(4), 375–398 (1993)
13. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), pp. 1271–1276 (2006)
14. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the approximation of instance level update and erasure in description logics. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 403–408 (2007)
15. Duschka, O.M., Genesereth, M.R.: Answering recursive queries using views. In: Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1997), pp. 109–116 (1997)
16. Duschka, O.M., Genesereth, M.R., Levy, A.Y.: Recursive query plans for data integration. J. of Logic Programming 43(1), 49–73 (2000)
17. Euzenat, J., Schwaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
18. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. J. of Intelligent Information Systems 8(2), 117–132 (1997)
19. Genereseth, M.R., Keller, A.M., Duschka, O.M.: Infomaster: An information integration system. In: ACM SIGMOD Int. Conf. on Management of Data, pp. 539–542 (1997)
20. Goh, C.H., Bressan, S., Madnick, S.E., Siegel, M.D.: Context interchange: New features and formalisms for the intelligent integration of information. ACM Trans. on Information Systems 17(3), 270–293 (1999)
21. Grieco, L., Lembo, D., Ruzzi, M., Rosati, R.: Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In: Conf. on Information and Knowledge Management (CIKM 2005), pp. 792–799 (2005)

22. Hull, R.: A survey of theoretical research on typed complex database objects. In: Paredaens, J. (ed.) Databases, pp. 193–256. Academic Press, London (1988)
23. Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The Information Manifold. In: Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments, pp. 85–91 (1995)
24. Kirsten, T., Rahm, E.: BioFuice: mapping-based data integration in bioinformatics. In: Leser, U., Naumann, F., Eckman, B. (eds.) DILS 2006. LNCS (LNBI), vol. 4075, pp. 124–135. Springer, Heidelberg (2006)
25. Lembo, D., Ruzzi, M.: Consistent query answering over description logic ontologies. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524. Springer, Heidelberg (2007)
26. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pp. 233–246 (2002)
27. Leone, N., Eiter, T., Faber, W., Fink, M., Gottlob, G., Greco, G., Kalka, E., Ianni, G., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Rosati, R., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX system for advanced integration of incomplete and inconsistent data. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 915–917 (2005)
28. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogenous information sources using source descriptions. In: Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB 1996) (1996)
29. Levy, A.Y., Srivastava, D., Kirk, T.: Data model and query evaluation in global information systems. J. of Intelligent Information Systems 5, 121–143 (1995)
30. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
31. Tomasic, A., Raschid, L., Valduriez, P.: Scaling access to heterogeneous data sources with DISCO. Trans. on Knowledge and Data Engineering 10(5), 808–823 (1998)
32. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)

The Relational Polynomial-Time Hierarchy and Second-Order Logic

Flavio A. Ferrarotti[1] and Jose M. Turull Torres[2]

[1] Information Systems, Department of Management
College of Business, Massey University
[2] School of Engineering and Advanced Technology
College of Sciences, Massey University PO Box 756, Wellington, New Zealand
{F.A.Ferrarotti,J.M.Turull}@massey.ac.nz

Abstract. We explore the connection between the concept of relational complexity introduced by S. Abiteboul, M. Vardi and V. Vianu and the restricted second-order logic SO^{ω} introduced by A. Dawar. In relational complexity, the basis for measuring the complexity of computing queries with relational machines, is the number of different FO^k-types realized by the input database. In SO^{ω}, the second-order quantifiers range over relations that are closed under equality of FO^k-types of k-tuples. We give a direct proof (in the style of the proof of Fagin's theorem) of the result of A. Dawar on the fact that the existential fragment of SO^{ω} captures *relational* NP. Then we define formally the concept of relational machine with *relational oracle* and show the exact correspondence between the prenex fragments of SO^{ω} and the levels of the *relational* polynomial-time hierarchy.

1 Introduction

Databases provide an abstract view of data purposely independent of the internal representation. Queries use only information provided by the abstract interface to the database. For instance, consider a database containing information on users of a web site (information such as pages visited, services requested, etc.), and a query which returns a subset of those users. The desired subset of users is specified by the query using only properties of the database. The users in the answer are *all* those which satisfy the properties specified in the query. Users which cannot be distinguished using those properties are treated uniformly. More formally, this is reflected in the concept of computable query introduced by Chandra and Harel [1], since it asks queries to be insensitive to isomorphisms of input structures.

Therefore, the usual notion of Turing machine computable function, which nowadays is the more widely accepted formalization of an effective procedure, is not suitable as formalization of database query. There is no formal restriction in the definition of Turing machine computation to force it to preserve isomorphisms of the input structures. In fact, if we want to compute queries using Turing machines, we need first to encode the database into a totally ordered structure which can then be written in the input tape.

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 48–76, 2008.

Several alternative abstract models of machines have been proposed to model the computation of queries to relational databases (see [2,3,4,5] among others). The common ground of all these devises is that they operate directly on structures rather than on encoding of structures as it is the case with Turing machines.

The fact that the models of computation of queries do not assume an ordered domain, leads to a mismatch between the hardness of database queries and their Turing complexity. The typical example is the *even* query on a set, which has very low Turing complexity, but it is by all accounts a hard query. This query cannot be expressed in logics as powerful as the infinitary logic with finitely many variables ($\mathcal{L}^{\omega}_{\infty\omega}$), which strictly includes all usual extensions of first-order logic with a fixed point operator. It is only when we assume a build in *order* that the inflationary fixed point logic (IFP) captures PTIME and the partial fixed point logic (PFP) captures PSPACE.

Based on this observation, S. Abiteboul, M. Vardi and V. Vianu [6] proposed to measure the complexity of computing queries using relational machines. This abstract model of machine was introduced in [3,4] and originally called loosely coupled generic machine. Latter on, it was renamed as relational machine [7,6]. This gave rise to the concept of *relational complexity* in which the number of different FO^k-types realized by the input database is used as the basis for measuring the complexity of computing queries with relational machines.

In [8], A. Dawar introduced a restricted version of second-order logic SO^{ω} in which the second-order quantifiers range over relations that are closed under equality of FO^k-types of k-tuples, i.e., closed under the equivalence relation $\equiv^k$ of k variable equivalence. Among other results, he proved that the existential fragment of SO^{ω} is equivalent to the nondeterministic inflationary fixed point logic (NFP). Since by a result of S. Abiteboul, M. Vardi and V. Vianu [6], NFP captures *relational* NP (denoted NP_r), it then follows that the existential fragment of SO^{ω} captures NP_r.

Continuing this line of work, we explore in detail the connection between the concept of relational complexity and the restricted second-order logic SO^{ω}. The aim is to provide the basis for a new line of research in the area of higher-order logics in finite models.

In this paper, we give a direct proof (in the style of the proof of Fagin's theorem) of the result of A. Dawar on the fact that the existential fragment of SO^{ω} captures NP_r. Then we define formally the concept of relational machine with *relational oracle* and show the exact correspondence between the prenex fragments of SO^{ω} and the levels of the *relational* polynomial-time hierarchy.

This last result is the *relational* equivalent of Stockmeyer characterization of the polynomial-time hierarchy [9]. It was already pointed out by A. Dawar in [8], though he did not prove it. He also observed that, if we close the logic NFP simultaneously under negation and the operation of taking nondeterministic fixed points, we obtain a logic equivalent to SO^{ω}. Moreover, the alternations of negations and fixed points correspond exactly to the second-order quantifier alternations in the prenex fragments of SO^{ω}.

However, up to our knowledge, this is the first attempt to formally define the *relational* polynomial-time hierarchy in terms of relational machines with *relational oracles*, i.e., oracles consisting on classes of structures closed under isomorphisms instead of sets of strings. This allows us to stablish a direct connection between the *relational* polynomial hierarchy and SO^{ω}, without using the S. Abiteboul and V. Vianu normal form for relational machines (see [4]). That is, we do not need to encode a canonical representation of the input structure in the Turing machine tape of the relational machine as in [6].

It is interesting to note that the idea behind SO^{ω} can be used to define restricted versions of higher-order logics of order higher than two. We believe that the correspondence between the *relational* polynomial hierarchy and the prenex fragments $\Sigma_m^{1,\omega}$ of SO^{ω}, can be extended to higher orders. We plan to pursue this matter further in the near future.

The paper is organized as follows. In the next section we present the technical background for the material which we cover in this article. This includes the necessary background on the model theoretic concept of *type* and the notion of *relational complexity* which play a central role in this work. In Section 3, we present the syntax and semantics of the logic SO^{ω} introduced by A. Dawar in [8]. We use this logic to characterize the *relational* polynomial-time hierarchy which we formally define in Section 4. Also in Section 4 we give a formal definition of relational machine and relational oracle. Finally in Section 5 we present our study of the relational complexity of SO^{ω}.

2 Preliminaries

As usual ([10,11]), we regard a *relational database schema*, as a relational vocabulary, and a *database instance* or simply *database* as a finite structure of the corresponding vocabulary. If $\mathbf{I}$ is a database or finite structure of some schema σ, we denote its domain as I and sometimes also as $dom(\mathbf{I})$. If R is a relation symbol in σ of arity r, for some $r \geq 1$, we denote as $R^{\mathbf{I}}$ the (second-order) relation of arity r which interprets the relation symbol R in $\mathbf{I}$, with the usual notion of interpretation. We denote as $\mathcal{B}_\sigma$ the class of databases of schema σ, or *finite* σ-structures.

In this paper, we consider *total* queries only. Let σ be a schema, let $r \geq 1$, and let R be a relation symbol of arity r. A *computable query of arity r and schema* σ ([1]), is a total recursive function $q : \mathcal{B}_\sigma \to \mathcal{B}_{\langle R \rangle}$ which preserves isomorphisms such that for every database $\mathbf{I}$ of schema σ, $dom(q(\mathbf{I})) \subseteq I$. A Boolean query is a 0-ary query.

We use the notion of a *logic* in a general sense. A formal definition would only complicate the presentation and is unnecessary for our work. As usual in finite model theory, we regard a logic as a language, that is, as a set of formulas (see [10]). We do require that the sintax and the notion of satisfaction for any logic $\mathcal{L}$ be *decidable*. We only consider vocabularies which are purely *relational*, and for simplicity we do not allow constant symbols. If σ is a relational vocabulary, we denote by $\mathcal{L}[\sigma]$ the set of $\mathcal{L}$-formulae over σ. We consider *finite* structures only.

Consequently, the notion of *satisfaction*, denoted as $\models$, is related to only finite structures.

By $\varphi(x_1, \ldots, x_r)$ we denote a formula of some logic whose free variables are *exactly* $\{x_1, \ldots, x_r\}$. If $\varphi(x_1, \ldots, x_r) \in \mathcal{L}[\sigma]$, $\mathbf{I} \in \mathcal{B}_\sigma$, $\bar{a}_r = (a_1, \ldots, a_r)$ is a r-tuple over $\mathbf{I}$, let $\mathbf{I} \models \varphi(x_1, \ldots, x_r)[a_1, \ldots, a_r]$ denote that φ is TRUE, when interpreted by $\mathbf{I}$, under a valuation v where for $1 \leq i \leq r$ $v(x_i) = a_i$. Then we consider the set of all such valuations as follows:

$$\varphi^{\mathbf{I}} = \{(a_1, \ldots, a_r) : a_1, \ldots, a_r \in I \wedge \mathbf{I} \models \varphi(x_1, \ldots, x_r)[a_1, \ldots, a_r]\}$$

That is, $\varphi^{\mathbf{I}}$ is the relation defined by φ in the structure $\mathbf{I}$, and its arity is given by the number of free variables in φ. Formally, we say that a formula $\varphi(x_1, \ldots, x_r)$ of signature σ, *expresses* a query q of schema σ, if for every database $\mathbf{I}$ of schema σ, is $q(\mathbf{I}) = \varphi^{\mathbf{I}}$. Similarly, a sentence φ expresses a Boolean query q if for every database $\mathbf{I}$ of schema σ, is $q(\mathbf{I}) = 1$ iff $\mathbf{I} \models \varphi$. For $\varphi \in \mathcal{L}[\sigma]$ we denote by $Mod(\varphi)$ the class of finite σ-structures $\mathbf{I}$ such that $\mathbf{I} \models \varphi$. A class of finite σ-structures $\mathcal{C}$ is *definable* by a $\mathcal{L}$-sentence if $\mathcal{C} = Mod(\varphi)$ for some $\varphi \in \mathcal{L}[\sigma]$.

We assume that the reader is familiar with first-order logic and with the following extensions of first-order logic with fixed point operators which are well known in finite model theory [10,12]: inflationary fixed point logic (IFP), least fixed point logic (LFP), and partial fixed point logic (PFP).

The fixed point logics IFP, LFP and PFP are obtained by deterministically iterating first-order operators. S. Abiteboul, M. Vardi and V. Vianu introduced in [6] a fixed point logic which is obtained by nondeterministically iterating first-order operators. Given two first-order formulae $\varphi_0(x_1, \ldots, x_k, R)$ and $\varphi_1(x_1, \ldots, x_k, R)$ of a same vocabulary σ, we define a sequence of stages $F_b^{(\varphi_0,\varphi_1)}$ indexed by binary strings $b \in \{0,1\}^*$, as follows:

$$F_\lambda^{(\varphi_0,\varphi_1)} = \emptyset, \text{ for the empty string } \lambda$$

$$F_{b\cdot 0}^{(\varphi_0,\varphi_1)} = F_b^{(\varphi_0,\varphi_1)} \cup F^{\varphi_0}(F_b^{(\varphi_0,\varphi_1)})$$

$$F_{b\cdot 1}^{(\varphi_0,\varphi_1)} = F_b^{(\varphi_0,\varphi_1)} \cup F^{\varphi_1}(F_b^{(\varphi_0,\varphi_1)}).$$

The *nondeterministic fixed point* of the sequence is $\bigcup_{b\in\{0,1\}^*} F_b^{(\varphi_0,\varphi_1)}$. The *nondeterministic inflationary fixed point logic* (NFP) is the closure of first-order logic under the operation of taking nondeterministic inflationary fixed points, with the restriction that negation cannot be applied to the fixed point operator.

Another way of extending the expressive power of first-order logic is by allowing disjunctions and conjunctions of arbitrary (not just finite) sets of formulae. But the resulting logic, usually denoted as $\mathcal{L}_{\infty\omega}$, is of little use in the study of finite models, since every query (including noncomputable queries) over finite structures is expressible in it. This changes when we concentrate on a bounded variable fragment of $\mathcal{L}_{\infty\omega}$ as all fixed point logics mentioned above can be viewed as fragments of it. More precisely, they can be viewed as fragments of the logic $\mathcal{L}^\omega_{\infty\omega} = \bigcup_{k\in\mathbb{N}} \mathcal{L}^k_{\infty\omega}$, where $\mathcal{L}^k_{\infty\omega}$ denotes the class of formulae of $\mathcal{L}_{\infty\omega}$ that use

at most k different variables. The following picture follows from the works of P. Kolaitis and M. Vardi [13,14] and A. Dawar [15].

$$\text{IFP} = \text{LFP} \subseteq \text{NFP} \subseteq \text{PFP} \subset \mathcal{L}^{\omega}_{\infty\omega}$$

The last containment is proper, since there are nonrecursive queries which can be expressed in $\mathcal{L}^{\omega}_{\infty\omega}$ [14] while every query definable in PFP is computable in PSPACE. In fact, as noted by A. Dawar [8], it can be shown that $\mathcal{L}^{k}_{\infty\omega}$ is complete on ordered structures, where the maximum arity of a relation symbol in its vocabulary is $\leq k$.

Second-order logic (SO) is yet another extension of first-order logic which allows to quantify over relations. In addition to the symbols of first-order logic, its alphabet contains, for each $n \geq 1$, countably many n-ary *relation variables*. As usual, we will use upper case letters to denote second-order relation variables. We define the set of second-order formulae of vocabulary σ to be the set generated by the rules for first-order formulas extended by:

- If X is a relation variable of arity n and $x_0, \ldots, x_{n-1}$ are individual variables, then $X(x_0, \ldots, x_{n-1})$ is a formula.
- If φ is a formula and X is a relation variable, then $\exists X(\varphi)$ and $\forall X(\varphi)$ are formulae.

The free occurrence of a relation variable in a second-order formula is defined in the obvious way and the notion of satisfaction is extended canonically. Then, the informal semantics of $\exists X(\varphi)$ and $\forall X(\varphi)$ over a relational structure $\mathbf{I}$, where X is a relation variable of arity r, is "There is at least one relation $R^{\mathbf{I}} \subseteq I^r$ such that φ is true when X is interpreted by $R^{\mathbf{I}}$" and "For every relation $R^{\mathbf{I}} \subseteq I^r$, φ is true when X is interpreted by $R^{\mathbf{I}}$", respectively. Given a σ-structure $\mathbf{I}$, a formula $\varphi = \varphi(x_0, \ldots, x_n, X_0, \ldots, X_k)$ with free individual variables among $x_0, \ldots, x_n$ and free relation variables among $X_0, \ldots X_k$, elements $a_0, \ldots, a_n \in I$, and relations $R^{\mathbf{I}}_0, \ldots, R^{\mathbf{I}}_k$, over $\mathbf{I}$ of arities corresponding to $X_0, \ldots, X_k$, respectivelly, we say that $\mathbf{I} \models \varphi[a_0, \ldots, a_n, R^{\mathbf{I}}_0, \ldots, R^{\mathbf{I}}_k]$ if the elements $a_0, \ldots, a_n$ together with $R^{\mathbf{I}}_0, \ldots, R^{\mathbf{I}}_k$ satisfy φ in $\mathbf{I}$.

It is a well known fact that every second-order formula is logically equivalent to one in prenex normal form in which each second-order quantifier precedes all first-order quantifiers. Such a formula is called Σ^1_m, if the string of second-order quantifiers consists of m consecutive blocks, where in each block all quantifiers are of the same type (i.e., all universal or all existential), adjacent blocks contain quantifiers of different type, and the first block is existential. Π^1_m is defined in the same way, but it is required that the first block consists of universal quantifiers.

2.1 Type of a Tuple

We need to consider the properties of a tuple in a relational structure or database, which are definable in a given logic. For this, we use the model theoretic concept of type. The results that we use in this work are mainly from [15,16]. Another excellent source for the subject is [17]. Our notation comes mostly from there.

Definition 1. *Let $\mathcal{L}$ be a logic, let $\mathbf{I}$ be a relational structure of vocabulary σ, and let $\bar{a} = (a_1, \ldots, a_k)$ be a k-tuple over $\mathbf{I}$. The* $\mathcal{L}$-type *of $\bar{a}$ in I, denoted $tp^{\mathcal{L}}_{\mathbf{I}}(\bar{a})$, is the set of formulas in $\mathcal{L}[\sigma]$ with free variables among $\{x_1, \ldots, x_k\}$ which are satisfied in $\mathbf{I}$ by any valuation which, for $1 \leq i \leq k$, assigns the i-th component of $\bar{a}$ to the variable x_i. In symbols,*

$$tp^{\mathcal{L}}_{\mathbf{I}}(\bar{a}) = \{\varphi \in \mathcal{L}[\sigma] : free(\varphi) \subseteq \{x_1, \ldots, x_k\} \text{ and } \mathbf{I} \models \varphi[a_1, \ldots, a_k]\}$$

Note that, the $\mathcal{L}$-type of a given tuple $\bar{a}$ over a relational structure $\mathbf{I}$, includes not only the properties of all sub-tuples of $\bar{a}$, but also the set of all sentences in $\mathcal{L}$ which are true when evaluated on $\mathbf{I}$, i.e., the $\mathcal{L}$-theory of $\mathbf{I}$.

According to Definition 1, a type is an *infinite* set of formulas which is *consistent*, i.e., there is a structure and a valuation which satisfy all the formulas in the set. Moreover, the set is *maximally consistent*, that is, if we add any formula to the set, we loose the consistency of the set. Therefore, we can think of the type of a tuple as a maximally consistent set of formulae, without having a particular relational structure in mind. If α is the $\mathcal{L}$-type of some tuple $\bar{a}$, i.e., α is a maximally consistent set of $\mathcal{L}$-formulae of some vocabulary σ, we say that a given σ-structure $\mathbf{I}$ *realizes* the type α if there is a tuple $\bar{a}$ over $\mathbf{I}$ such that $tp^{\mathcal{L}}_{\mathbf{I}}(\bar{a}) = \alpha$.

The notion of FO-type, usually encountered in classical (infinite) model theory, is not of much value in the context of finite model theory, since every tuple can be characterized up to isomorphism by its FO-type. However, if we consider logics which are weaker than first-order as to expressive power, that is not longer the case. In this context, a notion which has proven to be of great importance is that of FO^k-type, where for $k > 0$, FO^k denotes the fragment of first-order logic where only formulae with variables in $\{x_1, \ldots, x_k\}$ are allowed. Note that the class of queries expressible in FO^k is strictly included in the class of queries expressible in FO and that $\mathrm{FO} = \bigcup_{k>0} \mathrm{FO}^k$.

Definition 2. *For $k \geq r \geq 1$, we denote by $\equiv^k$ the* equivalence relation *induced in the set of r-tuples over a given structure $\boldsymbol{I}$, by equality of* FO^k*-types of r-tuples. That is, for every pair of r-tuples $\bar{a}$ and $\bar{b}$ over $\boldsymbol{I}$, $\bar{a} \equiv^k \bar{b}$ iff $tp^{\mathrm{FO}^k}_{\boldsymbol{I}}(\bar{a}) = tp^{\mathrm{FO}^k}_{\boldsymbol{I}}(\bar{b})$.*

P. Kolaitis and M. Vardi [14] showed that on finite structures, the equivalence relation $\equiv^k$ and the apparently stronger notion of equivalence of $\mathcal{L}^k_{\infty\omega}$-types, actually coincide. That is, they showed that if $\bar{a} \equiv^k \bar{b}$ over a given finite structure $\mathbf{I}$ of vocabulary σ, then for every $\varphi \in \mathcal{L}^k_{\infty\omega}[\sigma]$, it holds that $\mathbf{I} \models \varphi[\bar{a}]$ iff $\mathbf{I} \models \varphi[\bar{b}]$, and vice versa.

Definition 3. *Let $k \geq 1$, let σ be a relational vocabulary, and let $\boldsymbol{I}$ be a σ-structure. We say that a relation $R^{\mathbf{I}}$ of arity $r \leq k$ is* closed *under the equivalence relation $\equiv^k$ iff, for every pair of r-tuples $\bar{a}$ and $\bar{b}$ over $\boldsymbol{I}$, if $\bar{a} \in R^{\mathbf{I}}$ and $\bar{a} \equiv^k \bar{b}$, then $\bar{b} \in R^{\mathbf{I}}$.*

P. Kolaitis and M. Vardi [14] also showed that a query q of arity $r \leq k$ is expressible in $\mathcal{L}^k_{\infty\omega}$ iff it is closed under $\equiv^k$. The same is true for the fixed point

logics mentioned earlier, since the classes of queries expressible in those logics are strictly included in the class of queries expressible in $\mathcal{L}^{\omega}_{\infty\omega}$.

Although types are infinite sets of formulas, due to a result of A. Dawar [15], a *single* FO^k-formula is equivalent to the FO^k-type of a tuple over a given database. The equivalence holds for all databases of the same schema.

Lemma 1 (Corollary 2.18 **in [15]).** *For every schema σ, for every database* $\mathbf{I}$ *of schema σ, for every $k \geq 1$, for every $1 \leq r \leq k$, and for every r-tuple $\bar{a}$ over* $\mathbf{I}$*, there is an* FO^k*-formula $\alpha \in tp^{\mathrm{FO}^k}_{\mathbf{I}}(\bar{a})$ such that for any database* $\mathbf{J}$ *of schema σ and for every r-tuple $\bar{b}$ over* $\mathbf{J}$*,* $\mathbf{J} \models \alpha[\bar{b}]$ *iff $tp^{\mathrm{FO}^k}_{\mathbf{I}}(\bar{a}) = tp^{\mathrm{FO}^k}_{\mathbf{J}}(\bar{b})$.*

If an FO^k formula α satisfies the condition of Lemma 1, we say that α *isolates* the $tp^{\mathrm{FO}^k}_{\mathbf{I}}(\bar{a})$.

It is well known that the relation $\equiv^k$ is uniformly definable in LFP, or equivalently in IFP.

Theorem 1 ([13,16]). *For every relational vocabulary σ and every $k \geq 1$, there is a formula φ^k_σ of* IFP*, with $2k$ free variables, such that on any finite σ-structure* $\boldsymbol{I}$*, given two k-tuples $\bar{a}$ and $\bar{b}$ on* $\boldsymbol{I}$*,* $\boldsymbol{I} \models \varphi^k_\sigma[\bar{a}, \bar{b}]$ *iff $\bar{a} \equiv^k \bar{b}$.*

Moreover, a total ordering of the FO^k-types of a given vocabulary is also uniformly definable in LFP, or equivalently in IFP.

Theorem 2 ([4,16]). *For every relational vocabulary σ and every $k \geq 1$, there is a formula λ^k_σ of* IFP*, with $2k$ free variables, such that on any finite σ-structure* $\boldsymbol{I}$*, λ^k_σ defines a reflexive and transitive relation $\leq^k$ on k-tuples such that for every two k-tuples $\bar{a}$ and $\bar{b}$ on* $\boldsymbol{I}$*, either $\bar{a} \leq^k \bar{b}$ or $\bar{b} \leq^k \bar{a}$ and both $\bar{a} \leq^k \bar{b}$ and $\bar{b} \leq^k \bar{a}$ hold iff $\bar{a} \equiv^k \bar{b}$.*

That is, for every relational vocabulary σ and every $k \geq 1$, the corresponding IFP-formula λ^k_σ defines, on any finite σ-structure $\mathbf{I}$, a preorder such that the corresponding equivalence relation is $\equiv^k$. Thus, λ^k_σ can be seen as defining a total order on the equivalence classes of $\equiv^k$.

2.2 Background from Relational Complexity

A *relational machine* is a Turing machine augmented with a finite set of fixed-arity relations forming a *relational store* (rs). Designated relations contain initially the input database, and one specific relation holds the output at the end of the computation. A relational machine uses a finite set of first-order formulae to interact with the rs. Transitions have the form:

If the internal state of the machine is q,
the tape head is reading symbol x and
the first-order sentence φ evaluates to true in the rs,
then change state to q',
write symbol x' in the tape,
move the tape head one cell to the left/right and

replace the r-ary relation R by the relation defined by the first-order formula $\psi(x_1, \ldots, x_r)$ in the structure contained in the rs.

We give a formal definition of this machine in Section 4.

Each relational machine has an associated *arity*, which is the maximum number of variables which appear in any formula in its finite control.

Unlike Turing machines, relational machines have limited access to their input. Note that a k-ary relational machine can only access to its input through a fixed set of FO^k queries. In particular, relational machines cannot compute the size of their input structures in the general case. This is so because the discerning power of the relational machines is limited. More precisely, a relational machine of arity k cannot distinguish between tuples of elements of the input structure whose respective FO^k-types coincide.

Proposition 1 ([6]). *Let $1 \leq r \leq k$. For every pair of r-tuples $\bar{a}$ and $\bar{b}$ over a relational structure $\boldsymbol{I}$, $\bar{a} \equiv^k \bar{b}$ iff no k-ary relational machine can distinguish among $\bar{a}$ and $\bar{b}$ over $\boldsymbol{I}$.*

Therefore, computations of k-ary relational machines are determined by the equivalence classes of the relation $\equiv^k$. In fact, as shown by S. Abiteboul and V. Vianu in [4], relational machines are complete on ordered input structures (where all distinct tuples have different FO^k-type), but they collapse to first-order logic on unordered sets (where the number of equivalence classes in $\equiv^k$ is bounded by a constant independently of the size of the input structure).

Since k-ary relational machines cannot distinguish between tuples which are $\equiv^k$-equivalent, they cannot compute the size of their input structures. But, they can compute the number of $\equiv^k$-classes.

Proposition 2 ([6]). *Let the k-*size *of a structure $\boldsymbol{I}$, denoted $\mathrm{size}_k(\boldsymbol{I})$, be the number of $\equiv^k$-classes of k-tuples over $\boldsymbol{I}$. For each $k \geq 1$ and relational vocabulary σ, there is a deterministic relational machine M_σ of arity $2k$ that outputs on its Turing machine tape, for an input structure $\boldsymbol{I}$ of vocabulary σ, a string of length $size_k(\boldsymbol{I})$ in time polynomial in $size_k(\boldsymbol{I})$.*

Based on these facts, S. Abiteboul and V. Vianu proposed to use the k-size as a basis for measuring the complexity of relational machines. This is also the approach that we follow here.

We think of a relational machine M as an *acceptor* of a *relational language*, i.e., a class of structures of a relational vocabulary closed under isomorphisms. The relational language accepted by M, denoted $\mathcal{L}(M)$, is simply the set of input structures accepted by M. If M is deterministic, then the computation time of M on an input structure $\mathbf{I}$ is the number of transitions that M makes before accepting or rejecting $\mathbf{I}$, while the computation space is the number of tape cells scanned. If M is nondeterministic, then we only consider accepting computations. In that case, the computation time of M on an input structure $\mathbf{I}$ is the number of transitions in the shortest accepting computation of M on $\mathbf{I}$, while the computation space is the minimum number of tape cells scanned in any accepting computation of M on $\mathbf{I}$.

Definition 4. *Let $\mathcal{L}(M)$ be the relational language accepted by a halting relational machine M of arity k. Let t and s be functions on the natural numbers such that $t(n) \geq n+1$ and $s(n) \geq 1$. Then we say that:*

- $\mathcal{L}(M) \in \mathrm{DTIME}_r(t(n))$ *if M is deterministic and its computation time on any input structure $\mathbf{I}$ is bounded above by $t(size_k(\boldsymbol{I}))$;*
- $\mathcal{L}(M) \in \mathrm{NTIME}_r(t(n))$ *if M is nondeterministic and its computation time on any input structure $\mathbf{I}$ is bounded above by $t(size_k(\boldsymbol{I}))$;*
- $\mathcal{L}(M) \in \mathrm{DSPACE}_r(s(n))$ *if M is deterministic and its computation space on any input structure $\mathbf{I}$ is bounded above by $s(size_k(\boldsymbol{I}))$;*
- $\mathcal{L}(M) \in \mathrm{NSPACE}_r(s(n))$ *if M is nondeterministic and its computation space on any input structure $\mathbf{I}$ is bounded above by $s(size_k(\boldsymbol{I}))$.*

Mirroring the classical complexity classes, we define the class P_r of the relational languages decidable by relational machines working in polynomial-time in the k-size of their input structures as $\mathrm{P}_r = \bigcup_{c\in\mathbb{N}} \mathrm{DTIME}_r(n^c)$; and the class NP_r of the relational languages decidable by nondeterministic relational machine working in polynomial-time in the k-size of their input structures as $\mathrm{NP}_r = \bigcup_{c\in\mathbb{N}} \mathrm{NTIME}_r(n^c)$. The class PSPACE_r of relational languages decidable by relational machines working in polynomial-space in the k-size of their input structures is $\bigcup_{c\in\mathbb{N}} \mathrm{DSPACE}_r(n^c)$.

A logic $\mathcal{L}$ *captures* a relational complexity class $\mathcal{C}_r$ iff every class of relational structures definable in $\mathcal{L}$ is in $\mathcal{C}_r$ and vice versa.

S. Abiteboul, M. Vardi and V. Vianu proved the following results relating fixed point logics over finite structures and relational complexity classes.

Theorem 3 ([4,6])

- IFP *captures* P_r,
- NFP *captures* NP_r,
- PFP *captures* PSPACE_r.

Interestingly, questions about containments among the standard complexity classes translate to questions about containments among the *relational* complexity classes.

Theorem 4 ([6]). *Let $Class(Resource, Control, Bound)$ and $Class_r(Resource, Control, Bound)$ denote the classical complexity class and the relational complexity class, respectively, where Resource is either time or space, Control is either deterministic, nondeterministic, or alternating, and Bound is the bounding function or family of functions. Let F_1 and F_2 be polynomially closed sets of time/space constructible functions, and let $Resource_1$, $Resource_2$, $Control_1$, $Control_2$ be kinds of resources and controls, respectively. It holds that,*

$$Class(Resource_1, Control_1, F_1) \subseteq Class(Resource_2, Control_2, F_2)$$

if and only if

$$Class_r(Resource_1, Control_1, F_1) \subseteq Class_r(Resource_2, Control_2, F_2).$$

It follows that the known relationships between deterministic and nondeterministic complexity classes, also hold for relational complexity classes. For instance, the class of relational languages decidable by nondeterministic relational machines in polynomial space collapses to PSPACE_r. Also, open questions about standard complexity classes translate to questions about relational complexity classes. For example, $\mathrm{P} = \mathrm{NP}$ iff $\mathrm{P}_r = \mathrm{NP}_r$.

Note that, the well known Abiteboul-Vianu theorem, which reduces the problem of separating complexity classes P and PSPACE to separating the fixed point logics LFP and PFP over *unordered* structures, follows from Theorems 3 and 4.

3 A Restricted Second-Order Logic

Motivated by the study of the finite model theory of the infinitary logic with finitely many variables ($\mathcal{L}^{\omega}_{\infty\omega}$), A. Dawar defined in [8] a restricted version of second-order logic SO^{ω} which is contained within $\mathcal{L}^{\omega}_{\infty\omega}$. This is obtained by restricting the interpretation of the second-order quantifiers to relations closed under the equivalence relation $\equiv^k$, for some k (see Definitions 2 and 3).

We define next the syntax and semantics of SO^{ω}.

Definition 5. *In addition to the symbols of first-order logic, the alphabet of* SO^{ω} *contains, for each* $k \geq 1$, *a* second-order quantifier $\exists^k$ *and countably many* k*-ary* relation variables $V_1^k, V_2^k, \ldots$ *To denote relation variables we use letters* $X, Y, \ldots$.

Let $m \geq 1$ *and let* σ *be a relational vocabulary, we denote by* $\Sigma^{1,\omega}_m[\sigma]$ *the class of formulae of the form*

$$\exists^{k^1_1} X_{11} \ldots \exists^{k^1_{s_1}} X_{1s_1} \forall^{k^2_1} X_{21} \ldots \forall^{k^2_{s_2}} X_{2s_2} \ldots Q^{k^m_1} X_{m1} \ldots Q^{k^m_{s_m}} X_{ms_m}(\varphi),$$

where for $i, j \geq 1$ *we have* $s_i, k^i_j \geq 1$ *and* $arity(X_{ij}) \leq k^i_j$, Q *is either* $\exists$ *or* $\forall$, *depending on whether* m *is odd or even, respectively, and* φ *is a first-order formula of vocabulary* $\sigma \cup \{X_{11}, \ldots, X_{1s_1}, X_{21}, \ldots, X_{2s_2}, \ldots, X_{m1}, \ldots, X_{ms_m}\}$. *As usual,* $\forall^k X(\varphi)$ *abbreviates* $\neg\exists^k X(\neg\varphi)$.

Similarly, we denote by $\Pi^{1,\omega}_m[\sigma]$ *the class of formulae of the form*

$$\forall^{k^1_1} X_{11} \ldots \forall^{k^1_{s_1}} X_{1s_1} \exists^{k^2_1} X_{21} \ldots \exists^{k^2_{s_2}} X_{2s_2} \ldots Q^{k^m_1} X_{m1} \ldots Q^{k^m_{s_m}} X_{ms_m}(\varphi),$$

where for $i, j \geq 1$ *we have* $s_i, k^i_j \geq 1$ *and* $arity(X_{ij}) \leq k^i_j$, Q *is either* $\forall$ *or* $\exists$, *depending on whether* m *is odd or even, respectively, and* φ *is a first-order formula of vocabulary* $\sigma \cup \{X_{11}, \ldots, X_{1s_1}, X_{21}, \ldots, X_{2s_2}, \ldots, X_{m1}, \ldots, X_{ms_m}\}$.

The set of formulae of SO^{ω} *is then defined as the union of the set of first-order formulae with* $\bigcup_{m\geq 1} \Sigma^{1,\omega}_m$.

The notion of satisfaction *in* SO^{ω} *extends the notion of satisfaction in first-order with the following rule:*

- $\mathbf{I} \models \exists^k X(\varphi)$ *where* $k \geq 1$, X *is a relation variable of arity* $r \leq k$, φ *is a wff of vocabulary* $\sigma \cup \{X\}$ *and* $\boldsymbol{I}$ *is a* σ*-structure, iff there is an* $R \subseteq I^r$ *such that* R *is closed under the equivalence relation* $\equiv^k$ *in* $\boldsymbol{I}$, *and* $(\mathbf{I}, R) \models \varphi$. *Here* $(\mathbf{I}, R)$ *is the* $(\sigma \cup \{X\})$*-structure expanding* $\boldsymbol{I}$, *in which* X *is interpreted as* R.

Note that, for each $\Sigma^{1,\omega}_m$-formula $\varphi \equiv \exists^{k_1} X_1 \ldots Q^{k_s} X_s(\psi)$, there is a formula $\hat{\varphi} \in \Sigma^1_m$ which is equivalent to φ. The corresponding formula $\hat{\varphi}$ is simply,

$$\exists X_1 \ldots Q X_s \Big(\psi \wedge \bigwedge_{1 \leq i \leq s} \gamma^{k_i}(X_i) \Big),$$

where $\gamma^{k_i}(X_i)$ expresses that X_i is $\equiv^{k_i}$-closed. Since IFP $\subseteq \Sigma^1_1 \cap \Pi^1_1$, it clearly follows from Theorem 1 that γ^k is definable in Σ^1_m.

It is is important to mention that the restricted second-order logic SO^ω is not really restricted over ordered structures.

Theorem 5 ([8]). *On ordered structures, for every $m \geq 1$, $\Sigma^{1,\omega}_m = \Sigma^1_m$ and $\Pi^{1,\omega}_m = \Pi^1_m$.*

4 Relational Machines

We already introduced the relational machine in Section 2.2. But for the detail of the proofs, we need a formal definition.

Recall that the relational machine consists of a Turing machine augmented with a finite set of fixed-arity relations forming a *relational store* (*rs*). We assume that the Turing machine component of our relational machine consists of a *finite control* which has a finite set of internal states, plus a one-way infinite *tape* equipped with a read/write *tape head* which can move right or left.

Definition 6. *We formally define a* deterministic relational machine *as an eleven-tuple, $\langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \tau, \sigma, T, \Omega, \Phi \rangle$, where:*

1. *Q is the finite set of internal states;*
2. *Σ is the tape alphabet;*
3. *$\mathfrak{b} \in \Sigma$ is the symbol denoting blank;*
4. *$q_0 \in Q$ is the initial state;*
5. *$F \subseteq Q$ is the set of accepting final states;*
6. *τ is the vocabulary of the* rs*;*
7. *$\sigma \subset \tau$ is the vocabulary of the input structure;*
8. *$T \in \tau \setminus \sigma$ is the output relation;*
9. *Ω is a finite set of first-order sentences of vocabulary τ;*
10. *Φ is a finite set of first-order formulas of vocabulary τ, and*
11. *$\delta : Q \times \Sigma \times \Omega \to \Sigma \times Q \times \{R, L\} \times \Phi \times \tau$ is a partial function called the transition function.*

Transitions are based on:

i. the current state;
ii. the content of the current tape cell; and
iii. the answer to a Boolean first-order query evaluated on the τ-structure held in the *rs*.

Situations in which the transition function is undefined indicate that the computation must stop. Otherwise, the result of the transition function is interpreted as follows:

i. the first component is the symbol to be written on the scanned cell of the tape;
ii. the second component is the new state;
iii. the third component specifies the moves of the tape head: R means moving one cell to the right and L means moving one cell to the left;
iv. the fourth component specifies an n-ary ($n \geq 1$) first-order query φ to be evaluated on the τ-structure held in the rs; and
v. the fifth component is an n-ary relation symbol in τ, which specifies the n-ary relation in the rs to be replaced by the relation obtained from the evaluation of φ.

We can now introduce for relational machines the analogous to the concepts of configuration (also called instantaneous description or snapshot) and computation of Turing machines.

Definition 7. *Given a relational machine M, a* configuration *of M is a description of the whole status of the computation: it includes the contents of the tape, the position of the tape head, the current internal state of the finite control, and the contents of the relational store. Formally, a configuration of M is a 3-tuple $(q, w, \boldsymbol{A})$ where q is the current internal state of M, $w \in \Sigma^*\#\Sigma^*$ represents the current contents of the tape, and $\boldsymbol{A}$ is the current τ-structure held in the* rs*. The symbol "#" is supposed not to be in Σ, and marks the position of the tape head (by convention, the head scans the symbol immediately at the right of the "#"). All symbols in the infinite tape not appearing in w are assumed to be the particular symbol blank "$\mathfrak{b}$".*

The machine starts in the initial state, with the input in the designated relations of the relational store, and an empty tape.

Definition 8. *The* initial configuration *of a relational machine M on an input structure $\boldsymbol{I}$ of vocabulary σ is $(q_0, \#, \boldsymbol{A})$, where $\boldsymbol{A}$ is the τ-structure which extends $\boldsymbol{I}$ with an empty relation $R_i^{\boldsymbol{A}}$ for each relation symbol R_i in τ. The head is assumed to be in the left-most position of the tape.*

The fact of accepting an input is indicated by an accepting configuration.

Definition 9. *An* accepting configuration *is a configuration whose state is an accepting state.*

A computation of a relational machine can now be defined as a sequence of configurations:

Definition 10. *Given a relational machine M and an input structure $\boldsymbol{I}$, a* partial computation *of M on $\boldsymbol{I}$ is a (finite or infinite) sequence of configurations of M, in which each step from a configuration to the next obeys the transition*

function. A computation *is a partial computation which start with the initial configuration, and ends in a configuration in which no more steps can be performed. An* accepting computation *is a computation ending in an accepting configuration, and in this case the input structure* $\boldsymbol{I}$ *is* accepted.

Relational machines give rise to three types of computational devices. First, we can think of a relational machine M as an *acceptor* of a *relational language*, i.e., a class of structures closed under isomorphisms. In this case the *relational language accepted* by M, denoted $\mathcal{L}(M)$, is the set of input structures accepted by M. We can also think of a relational machine M as computing a *relational function* from input structures to output relations. The relational function computed by M is defined on the relational language accepted by M, and for each accepted input structure the value of the relational function is the relation held in the output relation T in the *rs* when the machine stops in an accepting state. Finally, we can think of a relational machine M as computing a *mixed function*, i.e., a function from structures to strings where the output is written on the machine's tape. Again the function is defined on the relational language accepted by M. For each accepted input structure the value of this function is the word which appears in the tape when the machine stops in an accepting state.

The arity of a relational machine is the maximum number of variables which appear in any formula in its finite control.

Definition 11. *Let* $M = \langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \tau, \sigma, T, \Omega, \Phi\rangle$ *be a relational machine, the* arity *of* M*, denoted as* $arity(M)$*, is* $max(\{|var(\varphi)| : \varphi \in \Omega \cup \Phi\})$.

4.1 Nondeterministic Relational Machines

In analogy with nondeterministic Turing machines, we can define nondeterministic relational machines.

Definition 12. *A* nondeterministic relational machine *is a eleven-tuple,* $\langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \sigma, \tau, T, \Omega, \Phi\rangle$*, where each component is as in the deterministic case, with the exception that the transition function is defined by*

$$\delta : Q \times \Sigma \times \Omega \to \mathcal{P}(\Sigma \times Q \times \{R, L\} \times \Phi \times \tau)$$

where, for any set A*,* $\mathcal{P}(A)$ *denotes the powerset of* A.

All the definitions and remarks made for the deterministic case about configurations and computations, apply in the same manner to the nondeterministic model. However, on a given input structure there is now not only one computation, but a set of possible computations. Acceptance for nondeterministic relational machines is therefore defined as follows.

Definition 13. *An input structure* $\boldsymbol{I}$ *is* accepted *by an nondeterministic relational machine* M *iff there exists a computation of* M *on* $\boldsymbol{I}$ *ending in an accepting configuration. We denote by* $\mathcal{L}(M)$ *the* relational language accepted *by* M*, i.e., the class of* σ*-structures accepted by* M.

4.2 Relational Oracle Machines

Definition 14. *A* relational oracle machine *is a relational machine with a distinguished set of relations in its rs, called oracle relations, and three distinguished states $q_?$, the query state, and q_{YES}, q_{NO}, the answer states.*

Similarly to the case of oracle Turing machines, the computation of an oracle relational machine requires that an oracle language be fixed previously to the computation. But, since we are working with relational machines, it is natural to think of a *relational oracle language*, i.e., a class of structures closed under isomorphisms, rather than a set of strings. Let $\mathcal{C}$ be an arbitrary class of structures of some vocabulary σ^o which is closed under isomorphisms. The computation of a relational oracle machine M with oracle $\mathcal{C}$ and distinguished set of oracle relation symbols σ^o, proceeds like in an ordinary relational machine, except for transitions from the query state. From the query state M transfers into the state q_{YES} if the relational structure of vocabulary σ^o formed by the domain of the input structure and the distinguished set of oracle relations currently held in the rs, belongs to $\mathcal{C}$; otherwise, M transfers into the state q_{NO}.

4.3 Relational Polynomial-Time Hierarchy

The time complexity of oracle relational machines is defined precisely in the same way as with ordinary relational machines. Each query step counts as one ordinary step. Thus if $\mathcal{C}$ is any deterministic or nondeterministic relational time complexity class and $\mathcal{A}$ is a relational language, we can define $\mathcal{C}^{\mathcal{A}}$ to be the class of all relational languages accepted by halting relational machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now an oracle $\mathcal{A}$.

Definition 15. *The levels of the* relational polynomial-time hierarchy *are defined as follows:*

- $\Delta_0^{\mathrm{P}_r} = \Sigma_0^{\mathrm{P}_r} = \Pi_0^{\mathrm{P}_r} = \mathrm{P}_r$
- *and for $m > 0$,*

$$\Delta_{m+1}^{\mathrm{P}_r} = \mathrm{P}_r^{\Sigma_m^{\mathrm{P}_r}} \qquad \Sigma_{m+1}^{\mathrm{P}_r} = \mathrm{NP}_r^{\Sigma_m^{\mathrm{P}_r}} \qquad \Pi_{m+1}^{\mathrm{P}_r} = \mathrm{coNP}_r^{\Sigma_m^{\mathrm{P}_r}}.$$

The relational complexity class PH_r *is the union of all relational complexity classes in the relational polynomial time hierarchy, i.e.,* $\mathrm{PH}_r = \bigcup_{m\in\mathbb{N}} \Sigma_m^{\mathrm{P}_r}$.

5 The Relational Complexity of SO$^\omega$

We know by the work of A. Dawar that the expressive power of the fragment $\Sigma_1^{1,\omega}$ of SO$^\omega$ equals the expressive power of the nondeterministic inflationary fixed point logic.

Theorem 6 ([8]). $\Sigma_1^{1,\omega} = \mathrm{NFP}$.

It clearly follows from this result and Theorem 3 that the classes of relational structures which are finitely axiomatizable in $\Sigma_1^{1,\omega}$, are exactly those classes which belong to the relational complexity class NP_r.

Theorem 7 ([8]). *$\Sigma_1^{1,\omega}$ captures NP_r.*

We give next a direct proof of this result. That is, we show that for every relational vocabulary σ, every $\Sigma_1^{1,\omega}[\sigma]$-sentence φ can be evaluated in NP_r, and vice versa that every NP_r property of finite relational structures can be expressed in $\Sigma_1^{1,\omega}$. But first, we need some preparation.

The next lemma is a direct consequence of Theorems 3 and 2.

Lemma 2. *For every relational vocabulary σ and every $k \geq 1$, there is a deterministic relational machine $M_{\leq^k}$ of arity $k' \geq 2k$, such that on any input structure $\boldsymbol{I}$ of vocabulary σ, $M_{\leq^k}$ computes the preorder $\leq^k$ of Theorem 2 working in time bounded by a polynomial in $size_{k'}(\boldsymbol{I})$.*

The following two facts are variations of Facts 3.2 and 3.1 in [18], respectively. Anyway, we prove them for their better understanding.

Fact 8. *Let $\mathbf{I}$ be a relational structure of some vocabulary σ, and let $\bar{a} = (a_1, \ldots, a_k)$ and $\bar{b} = (b_1, \ldots, b_k)$ be two k-tuples on $\mathbf{I}$. Let $1 \leq r \leq k$ and let $(i_1, \ldots i_r)$ be a tuple of projection coordinates, where for $1 \leq j < r$, we have $1 \leq i_j < i_{j+1} \leq k$. Let $\bar{a}' = (a_{i_1}, \ldots, a_{i_r})$ and $\bar{b}' = (b_{i_1}, \ldots, b_{i_r})$. It follows that, if $\bar{a} \equiv^k \bar{b}$, then both $\bar{a}' \equiv^k \bar{b}'$ and $\bar{a}' \equiv^r \bar{b}'$ hold.*

Proof. By definition of the FO^k-type of a tuple, for every FO^k formula $\varphi \in tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a})$, with $free(\varphi) \subseteq \{x_{i_1}, \ldots, x_{i_r}\}$, it holds that $\mathbf{I} \models \varphi[\bar{a}]$. Since the valuation represented by the sub-tuple $\bar{a}' = (a_{i_1}, \ldots, a_{i_r})$ assigns to the free variables in φ the same elements from I as the valuation represented by the tuple $\bar{a}$, it also holds that $\mathbf{I} \models \varphi[\bar{a}']$. Thus we can define $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a}')$ as the set of FO^k formulae $\varphi \in tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a})$ such that $free(\varphi) \subseteq \{x_{i_1}, \ldots, x_{i_r}\}$. Given that $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{b}')$ can be defined in the same way, if $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a}) = tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{b})$, then $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a}') = tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{b}')$.

To prove that $\bar{a}' \equiv^r \bar{b}'$, we use the same argument, except that we consider FO^r formulae $\varphi \in tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a})$ and FO^r formulae $\varphi \in tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{b})$, instead of FO^k formulae, to build $tp_{\mathbf{I}}^{\mathrm{FO}^r}(\bar{a}')$ and $tp_{\mathbf{I}}^{\mathrm{FO}^r}(\bar{b}')$, respectively. □

Fact 9. *Let $\mathbf{I}$ be a relational structure of some vocabulary σ and let φ be a $\mathrm{FO}^k[\sigma]$-formula with $1 \leq r \leq k$ free variables. The relation that φ defines on $\mathbf{I}$, i.e. $\varphi^{\mathbf{I}}$, is closed under $\equiv^k$.*

Proof. Let $\bar{a}$ and $\bar{b}$ be two r-tuples on $\mathbf{I}$. We show that, if $\bar{a} \in \varphi^{\mathbf{I}}$ and $\bar{a} \equiv^k \bar{b}$, then $\bar{b} \in \varphi^{\mathbf{I}}$. Since $\bar{a} \in \varphi^I$, then by definition of the FO^k-type of a tuple and definition of $\varphi^{\mathbf{I}}$, we have that φ is in $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a})$. Given that $\bar{a} \equiv^k \bar{b}$, φ is also in $tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{b})$, and therefore $I \models \varphi[\bar{b}]$. Hence, $\bar{b} \in \varphi^{\mathbf{I}}$. □

The following is a straightforward consequence of Fact 8.

Fact 10. *Let* $\mathbf{I}$ *be a relational structure of some vocabulary* σ*. For every* $1 \leq r \leq k$*, it holds that* $size_r(\mathbf{I}) \leq size_k(\mathbf{I})$*.*

We can now proceed with the first part of our direct proof of A. Dawar's result regarding the relational complexity of $\Sigma_1^{1,\omega}$, i.e., Theorem 7.

Proposition 3. *Every class of relational structures definable in* $\Sigma_1^{1,\omega}$ *is in* NP_r*.*

Proof. We show that for every relational vocabulary σ, every $\Sigma_1^{1,\omega}[\sigma]$-sentence φ can be evaluated in NP_r. Suppose that φ is $\exists^{k_1} X_1 \ldots \exists^{k_s} X_s(\psi)$, where ψ is a first-order formula of vocabulary $\sigma \cup \{X_1, \ldots, X_s\}$. We build a nondeterministic relational machine M_φ which evaluates φ on input structures of vocabulary σ. For $1 \leq i \leq s$, let $k_i' \geq 2k_i$ be the arity of the relational machine $M_{\leq^{k_i}}$ of Lemma 2 which computes the preorder $\leq^{k_i}$ of Theorem 2. The arity k of M_φ is $max(\{k_1', \ldots, k_s'\})$. The vocabulary τ of the relational store is $\sigma \cup \{X_1, \ldots, X_s, \leq^{k_1}, \ldots, \leq^{k_s}, S_1, \ldots, S_s\}$, where for $1 \leq i \leq s$, the arity of X_i is exactly the same as the arity $r_i \leq k_i$ of the corresponding quantified relation variable in φ, the arity of $\leq^{k_i}$ is $2k_i$, and the arity of S_i is k_i. Let $\mathbf{I}$ be the input structure to M_φ. The machine works as follows:

- For each $1 \leq i \leq s$, M_φ builds the preorder $\leq^{k_i}$ of Theorem 2 in its rs. To complete this step, M_φ simply emulates, for each preorder $\leq^{k_i}$, the corresponding deterministic relational machine $M_{\leq^{k_i}}$ of arity $k_i' \geq 2k_i$ of Lemma 2. Therefore, M_φ can compute each preorder $\leq^{k_i}$ working deterministically in time bounded by a polynomial in $size_{k_i'}(\mathbf{I})$, and as $k \geq k_i'$, all preorders $\leq^{k_i}$ working deterministically in time bounded by a polynomial in $size_k(\mathbf{I})$ (which by Fact 10 is $\geq size_{k_i'}(\mathbf{I})$).
- By stepping through the equivalence classes of the relation $\equiv^{k_i}$ in the order given by $\leq^{k_i}$, M_φ computes $size_{k_i}(\mathbf{I})$ for every $1 \leq i \leq s$. Clearly the computation can be carried out by working deterministically in a number of steps polynomial in $size_k(\mathbf{I})$. See also Proposition 2.
- For $1 \leq i \leq s$, M_φ guesses and writes over its Turing machine tape a tuple $\bar{a}_i = (a_{i1}, \ldots, a_{i size_{k_i}(\mathbf{I})}) \in \{0,1\}^{size_{k_i}(\mathbf{I})}$. Since for each $1 \leq i \leq s$, M_φ can perform this task working nondeterministically in time $size_{k_i}(\mathbf{I})$ (which by Fact 10 is $\leq size_k(\mathbf{I})$), this computation takes time polynomial in $size_k(\mathbf{I})$.
- Using the binary tuples guessed in the previous step, M_φ generates, for every $1 \leq i \leq s$, a relation which is placed in X_i in its rs and is closed under the equivalence relation $\equiv^{k_i}$ in $\mathbf{I}$.

For $i = 1$ to s.
  For $j = 1$ to $size_{k_i}(\mathbf{I})$
    Begin
    If $a_{ij} = 1$ then
      $X_i := X_i(x_1, \ldots, x_{r_i}) \vee (\exists x_{r_i+1} \ldots x_{k_i}(\neg S_i(x_1, \ldots, x_{k_i}) \wedge$
        $\forall y_1 \ldots y_{k_i}(\leq^{k_i} (y_1, \ldots, y_{k_i}, x_1, \ldots, x_{k_i}) \wedge$
        $\neg \leq^{k_i} (x_1, \ldots, x_{k_i}, y_1, \ldots, y_{k_i}) \rightarrow S_i(y_1, \ldots y_{k_i}))));$
    $S_i := S_i(\bar{x}) \vee (\neg S_i(\bar{x}) \wedge \forall \bar{y}(\leq^{k_i} (\bar{y}, \bar{x}) \wedge \neg \leq^{k_i} (\bar{x}, \bar{y}) \rightarrow S_i(\bar{y})));$
    End;

M_φ can clearly perform this task working deterministically in time bounded by a polynomial in $size_k(\mathbf{I})$.
- Finally, M_φ evaluates ψ on the τ-structure currently held in its rs. M_φ accepts the input structure $\mathbf{I}$ iff ψ evaluates to true.

Hence, φ can be evaluated in NP_r. □

Before proving the converse of the previous result, we need an additional fact.

Fact 11. *Let $\mathbf{I}$ be a relational structure. For $1 \leq i \leq n$, let $k_i \geq r_i \geq 1$ and let $\mathcal{C}_i$ be the set of equivalence classes of r_i-tuples determined by $\equiv^{k_i}$ on $\mathbf{I}$. If follows that, for every $\mathcal{C}_R \subseteq \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$, the relation*

$$R^{\mathbf{I}} = \{(a_{11}, \ldots, a_{1r_1}, a_{21}, \ldots, a_{2r_2}, \ldots, a_{n1}, \ldots a_{nr_n}) \in I^{r_1+r_2+\ldots+r_n} :$$

$$([(a_{11}, \ldots, a_{1r_1})], [(a_{21}, \ldots, a_{2r_2})], \ldots, [(a_{n1}, \ldots a_{nr_n})]) \in \mathcal{C}_R\}$$

is closed under $\equiv^{k_1+k_2+\cdots+k_n}$ on $\mathbf{I}$.

Proof. Suppose that there is a $\mathcal{C}_R \subseteq \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$ such that its corresponding relation $R^{\mathbf{I}}$ is not closed under $\equiv^{k_1+k_2+\cdots+k_n}$ on $\mathbf{I}$. Then, there are two $(r_1 + r_2 + \cdots + r_n)$-tuples

$$\bar{a} = (a_{11}, \ldots, a_{1r_1}, a_{21}, \ldots, a_{2r_2}, \ldots, a_{n1}, \ldots, a_{nr_n}), \text{ and}$$

$$\bar{b} = (b_{11}, \ldots, b_{1r_1}, b_{21}, \ldots, b_{2r_2}, \ldots, b_{n1}, \ldots, b_{nr_n})$$

on $\mathbf{I}$ such that $\bar{a} \in R^{\mathbf{I}}$, $\bar{b} \notin R^{\mathbf{I}}$ and $\bar{a} \equiv^{k_1+k_2+\cdots+k_n} \bar{b}$. Since $\bar{a} \equiv^{k_1+k_2+\cdots+k_n} \bar{b}$, for $1 \leq i \leq n$, it holds that $(a_{i1}, \ldots, a_{ir_i}) \equiv^{k_i} (b_{i1}, \ldots, b_{ir_i})$. But then, by definition of $R^{\mathbf{I}}$, we have that $\bar{b} \in R^{\mathbf{I}}$ which contradicts our hypothesis. □

Fact 12. *Let M be a relational machine of arity k and let σ and τ be the vocabularies of the input structure and the rs, respectively. In every configuration in a computation of M on an input structure $\mathbf{I}$, every relation R_i of arity $r_i \leq k$ held in the rs of M is closed under the equivalence relation $\equiv^k$ on $\mathbf{I}$.*

Proof. We proceed by induction on the sequence of configurations of a computation of M on an input structure $\mathbf{I}$. W.l.o.g. we can assume that the relations of the input structure $\mathbf{I}$, i.e., the relations in the rs which interpret the relation symbols in σ, are not modified through any computation of M. This poses no problem since in case that such a modification is needed, we can just get another copy of the relation which would be in $\sigma \setminus \tau$ and modify the new relation instead.

In the initial configuration of M, the τ-structure $\mathbf{A}_0$ held in the rs is $\mathbf{I}$ extended with an empty relation $R_i^{\mathbf{A}_0}$ for each relation symbol R_i in $\tau \setminus \sigma$. Note that, by Fact 8, if a relation R of arity $r \leq k$ is closed under $\equiv^r$, then R is also closed under $\equiv^k$, hence we only need to show that for every relation symbol $R_i \in \sigma$ of arity r_i, the relation $R_i^{\mathbf{I}}$ is closed under $\equiv^{r_i}$ on $\mathbf{I}$. Let us assume that there is a $R_i \in \sigma$ such that $R_i^{\mathbf{I}}$ is not closed under $\equiv^{r_i}$ on $\mathbf{I}$. If that is the case, there are two r_i-tuples $\bar{a}, \bar{b} \in I^{r_i}$ such that $\bar{a} \in R_i^{\mathbf{I}}$, $\bar{b} \notin R_i^{\mathbf{I}}$ and $\bar{a} \equiv^{r_i} \bar{b}$. But

then there is a FO^{r_i}-formula of vocabulary σ, namely $R_i(x_1, \ldots, x_{r_i})$, such that $\mathbf{I} \models R_i(x_1, \ldots, x_{r_i})[\bar{a}]$ while $\mathbf{I} \not\models R_i(x_1, \ldots, x_{r_i})[\bar{b}]$, which clearly contradicts our assumption that $\bar{a} \equiv^{r_i} \bar{b}$.

For a configuration other than the initial configuration, we assume as inductive hypothesis that, in the preceding configuration in the sequence, every relation $R_i^{\mathbf{A}_{n-1}}$ of arity $r_i \leq k$ in the τ-structure $\mathbf{A}_{n-1}$ held in the rs of M, is closed under $\equiv^k$ on $\mathbf{I}$. Let $\mathbf{A}_n$ be the τ-structure held in the rs of M in the current configuration. Note that in each step from a configuration to the next M updates exactly one relation in its rs. Let $R_x^{\mathbf{A}_n}$ of arity r_x be the relation updated in the step from the previous to the current configuration in the sequence, and let $\varphi_{R_x} \in \mathrm{FO}^k[\tau]$ be the formula used for the update. We show below that there is a formula $\varphi'_{R_x} \in \mathrm{FO}^k[\sigma]$ such that $\varphi_{R_x}^{\mathbf{A}_{n-1}} = \varphi'^{\mathbf{I}}_{R_x}$, i.e., φ'_{R_x} defines in $\mathbf{I}$ the same relation that φ_{R_x} defines in $\mathbf{A}_{n-1}$. Since the FO^k-formula φ'_{R_x} of vocabulary σ defines on $\mathbf{I}$ the relation $R_x^{\mathbf{A}_n}$, it follows by Fact 9 that $R_x^{\mathbf{A}_n}$ is closed under $\equiv^k$ on $\mathbf{I}$.

By inductive hypothesis, if $R_i^{\mathbf{A}_{n-1}}$ is a relation of arity r_i in $\mathbf{A}_{n-1}$, then it is closed under $\equiv^k$ on $\mathbf{I}$. Let $T_{R_i} = \{tp_{\mathbf{I}}^{\mathrm{FO}^k}(\bar{a}) : \bar{a} \in R_i^{\mathbf{A}_{n-1}}\}$ be the set of FO^k-types realized by $R_i^{\mathbf{A}_{n-1}}$ on $\mathbf{I}$. By Lemma 1, for every type $t_j \in T_{R_i}$, there is an FO^k formula α_{t_j} of vocabulary σ which isolates t_j. Therefore, the FO^k-formula $\psi_{R_i} \equiv \bigvee_{t_j \in T_{R_i}} \alpha_{t_j}$ of vocabulary σ defines $R_i^{\mathbf{A}_{n-1}}$ on $\mathbf{I}$. The formula φ'_{R_x} is built from φ_{R_x} with every occurrence of a sub-formula of the form $R_i(x_1, \ldots, x_{r_i})$, where $R_i \in \tau \setminus \sigma$, replaced by the corresponding sub-formula $\psi_{R_i}(x_1, \ldots, x_{r_i})$. □

We complete the proof of Theorem 7 showing that every NP_r property of finite relational structures can be expressed in $\Sigma_1^{1,\omega}$. The proof is close to the proof of Fagin's theorem in [12], but we need to bear in mind that we can only quantify relational variables which are closed under the equivalence relation $\equiv^k$ for some k, and we have to take into account the rs of the machine.

Proposition 4. *Every class of relational structures (relational language) in* NP_r *is definable in* $\Sigma_1^{1,\omega}$.

Proof. Let $q : \mathcal{B}_\sigma \to \{0, 1\}$ be a Boolean query which is computed by a nondeterministic relational machine $M = \langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \sigma, \tau, R_l, \Omega, \Phi \rangle$ of arity k, working in polynomial time in the k-size of the input structure of vocabulary σ. We assume that M works in time $(size_k(\mathbf{I}))^s$ for some $s \geq 1$ and $\mathbf{I} \in \mathcal{B}_\sigma$. Here

- $Q = \{q_0, \ldots, q_m\}$ is the finite set of internal states; $q_0 \in Q$ is the initial state;
- $\Sigma = \{0, 1, \mathfrak{b}\}$ is the tape alphabet; $\mathfrak{b} \in \Sigma$ is the symbol denoting blank;
- $F = \{q_m\}$ is the set of accepting final states;
- $\tau = \{R_0, \ldots, R_l\}$ is the vocabulary of the rs, where for $0 \leq i \leq l$, the arity of R_i is r_i;
- $\sigma = \{R_0, \ldots, R_u\}$, where $u < l$, is the vocabulary of the input structure;
- R_l is the output relation;
- $\Omega = \{\alpha_0, \ldots, \alpha_v\}$ is a finite set of first-order sentences of vocabulary τ;
- $\Phi = \{\gamma_0, \ldots, \gamma_w\}$ is a finite set of first-order formulas of vocabulary τ; and
- $\delta : Q \times \Sigma \times \Omega \to \mathcal{P}(\Sigma \times Q \times \{R, L\} \times \Phi \times \tau)$ is the transition function of M.

The sentence φ_M expressing acceptance by M on input $\mathbf{I} \in \mathcal{B}_\sigma$ has the form

$$\exists^{2\cdot k} O\, \exists^{2\cdot s\cdot k} T_0\, \exists^{2\cdot s\cdot k} T_1\, \exists^{2\cdot s\cdot k} T_{\mathfrak{b}}\, \exists^{2\cdot s\cdot k} H_{q_0} \ldots \exists^{2\cdot s\cdot k} H_{q_m}\, \exists^{(s+1)\cdot k} S_0 \ldots \exists^{(s+1)\cdot k} S_l\, (\psi),$$

where ψ is a first-order formula of vocabulary $\sigma \cup \{O, T_0, T_1, T_{\mathfrak{b}}, H_{q_0}, \ldots, H_{q_m}, S_0, \ldots, S_l\}$. The arity of O is $2 \cdot k$; the arity of $T_0, T_1, T_{\mathfrak{b}}$ as well as the arity of H_{q_i} for $0 \leq i \leq m$, is $2 \cdot s \cdot k$; and the arity of S_i for $0 \leq i \leq l$ is $r_i + s \cdot k$. The intended interpretation of these relation symbols is as follows:

– O is a preorder of k-tuples over $\mathbf{I}$.

Note that, by Theorem 2, there is a preorder $\leq^k$ such that the corresponding equivalence relation is $\equiv^k$, and that such preorder is also a linear order over the set of equivalence classes of k-tuples $\mathcal{C}$ determined by $\equiv^k$. Thus, with $\leq^k$ we can define a lexicographic linear order of the s-tuples in $\mathcal{C}^s$. Since M runs in time bounded by $(size_k(\mathbf{I}))^s$ and visits at most $(size_k(\mathbf{I}))^s$ cells, we can model time $(\bar{t})$ as well as position on the tape $(\bar{p})$ by s-tuples of equivalence classes in $\mathcal{C}$. We actually do that by using $(s \cdot k)$-tuples of elements of the domain I instead of s-tuples of equivalence classes in $\mathcal{C}$. Under this approach, two $(s \cdot k)$-tuples $\bar{a} = (\bar{a}_1, \ldots, \bar{a}_s)$ and $\bar{b} = (\bar{b}_1, \ldots, \bar{b}_s)$ are considered equivalent iff, for $1 \leq i \leq s$, their corresponding k-tuples $\bar{a}_i$ and $\bar{b}_i$ belong to the same equivalence class, i.e., iff $\bar{a}_i \leq^k \bar{b}_i$ and $\bar{b}_i \leq^k \bar{a}_i$.

Having this considerations in mind, we define the intended interpretation of the remaining relation symbols as follows:

– T_0, T_1, and $T_{\mathfrak{b}}$ are *tape* relations; for $x \in \{0, 1, \mathfrak{b}\}$, $T_x(\bar{p}, \bar{t})$ indicates that position $\bar{p}$ at time $\bar{t}$ contains x.
– H_q's are *head* relations; for $q \in Q$, $H_q(\bar{p}, \bar{t})$ indicates that at time $\bar{t}$, the machine M is in state q, and its head is in position $\bar{p}$.
– S_i's are rs relations; for $0 \leq i \leq l$, $S_i(\bar{a}, \bar{t})$ indicates that at time $\bar{t}$, the relation R_i in the rs contains the r_i-tuple $\bar{a}$.

The sentence ψ must now express that when M starts with an empty tape and an input $\mathbf{I}$ in the designated relations of its rs, the relations T_x's, H_q's and S_i's encode its computation, and eventually M reaches an accepting state.

We define ψ to be the conjunction of the following sentences:

– A sentence expressing that O defines a pre-order of k-tuples.
  $\forall \bar{x}(O(\bar{x}, \bar{x}))$ "O is reflexive" $\wedge$
  $\forall \bar{x}\bar{y}\bar{z}(O(\bar{x}, \bar{y}) \wedge O(\bar{y}, \bar{z}) \rightarrow O(\bar{x}, \bar{z}))$ "O is transitive" $\wedge$
  $\forall \bar{x}\bar{y}(O(\bar{x}, \bar{y}) \vee O(\bar{y}, \bar{x}))$ "O is connex".
– A sentence defining the initial configuration of M.
  $\exists \bar{x} \forall \bar{y} \Big(\bar{x} \preceq \bar{y}) \rightarrow \big(H_{q_0}(\bar{x}, \bar{x}) \wedge \forall \bar{p}(T_{\mathfrak{b}}(\bar{p}, \bar{x}))\big)\Big)$
  "At time 0, M is in state q_0, the head is in the left-most position of the tape, and the tape contains only blanks" $\wedge$

$$\exists \bar{t} \forall \bar{x} \Big(\bar{t} \preceq \bar{x} \rightarrow \bigwedge_{0 \leq i \leq u} \big(\forall a_1 \ldots a_{r_i} (S_i(a_1, \ldots, a_{r_i}, \bar{t}) \leftrightarrow R_i(a_1, \ldots, a_{r_i})) \big) \wedge \bigwedge_{u < i \leq l} \big(\forall a_1 \ldots a_{r_i} (\neg S_i(a_1, \ldots, a_{r_i}, \bar{t})) \big) \Big)$$

"the relations in the rs hold a τ-structure $\mathbf{A}$ which extends $\mathbf{I}$ with an empty relation $S_i^{\mathbf{A}}$ for each relation symbol R_i in $\tau \setminus \sigma$".

Here, for $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_s)$ and $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_s)$, where $\bar{x}_i$ and $\bar{y}_i$ $(1 \leq i \leq s)$ are k-tuples of individual variables, we say that $\bar{x} \preceq \bar{y}$ iff

$$\big(O(\bar{x}_1, \bar{y}_1) \wedge \neg O(\bar{y}_1, \bar{x}_1) \big) \vee$$
$$\big(O(\bar{x}_1, \bar{y}_1) \wedge O(\bar{y}_1, \bar{x}_1) \wedge O(\bar{x}_2, \bar{y}_2) \wedge \neg O(\bar{y}_2, \bar{x}_2) \big) \vee \ldots \vee$$
$$\big(O(\bar{x}_1, \bar{y}_1) \wedge O(\bar{y}_1, \bar{x}_1) \wedge \ldots \wedge O(\bar{x}_{s-1}, \bar{y}_{s-1}) \wedge O(\bar{y}_{s-1}, \bar{x}_{s-1}) \wedge$$
$$O(\bar{x}_s, \bar{y}_s) \wedge \neg O(\bar{y}_s, \bar{x}_s) \big) \vee \bar{x} \sim \bar{y},$$

where $\bar{x} \sim \bar{y}$ is simply $O(\bar{x}_1, \bar{y}_1) \wedge O(\bar{y}_1, \bar{x}_1) \wedge \ldots \wedge O(\bar{x}_s, \bar{y}_s) \wedge O(\bar{y}_s, \bar{x}_s)$. Informally, $\bar{x} \preceq \bar{y}$ if $\bar{x}$ precedes or equals $\bar{y}$ in the lexicographic order induced by O, and $\bar{x} \sim \bar{y}$ if they share the same position.

– A sentence stating that in every configuration of M, each cell of the tape contains exactly one element of Σ.

$$\forall \bar{p} \bar{t} \Big(\big(T_0(\bar{p}, \bar{t}) \leftrightarrow \forall \bar{x} \bar{y} (\bar{x} \sim \bar{p} \wedge \bar{y} \sim \bar{t} \rightarrow T_0(\bar{x}, \bar{y}) \wedge \neg T_1(\bar{x}, \bar{y}) \wedge \neg T_{\mathfrak{b}}(\bar{x}, \bar{y})) \big) \wedge$$
$$\big(T_1(\bar{p}, \bar{t}) \leftrightarrow \forall \bar{x} \bar{y} (\bar{x} \sim \bar{p} \wedge \bar{y} \sim \bar{t} \rightarrow T_1(\bar{x}, \bar{y}) \wedge \neg T_0(\bar{x}, \bar{y}) \wedge \neg T_{\mathfrak{b}}(\bar{x}, \bar{y})) \big) \wedge$$
$$\big(T_{\mathfrak{b}}(\bar{p}, \bar{t}) \leftrightarrow \forall \bar{x} \bar{y} (\bar{x} \sim \bar{p} \wedge \bar{y} \sim \bar{t} \rightarrow T_{\mathfrak{b}}(\bar{x}, \bar{y}) \wedge \neg T_0(\bar{x}, \bar{y}) \wedge \neg T_1(\bar{x}, \bar{y})) \big) \Big).$$

– A sentence stating that at any time the machine is in exactly one state.

$$\forall \bar{t} \exists \bar{p} \Big(\bigvee_{q \in Q} \big(H_q(\bar{p}, \bar{t}) \wedge \forall \bar{x} (H_q(\bar{x}, \bar{t}) \leftrightarrow \bar{x} \sim \bar{p}) \big) \Big) \wedge$$
$$\neg \exists \bar{p} \bar{t} \bar{x} \bar{y} \Big(\bigvee_{q, q' \in Q, q \neq q'} \big(H_q(\bar{p}, \bar{t}) \wedge \bar{p} \sim \bar{x} \wedge \bar{t} \sim \bar{y} \wedge H_{q'}(\bar{x}, \bar{y}) \big) \Big).$$

– Sentences expressing that the relations T_i's, H_q's and S_i's respect the transitions of M. For every $a \in \Sigma$, $q \in Q$ and $\alpha \in \Omega$ for which the transition function δ is defined, we have a sentence of the form

$$\bigvee_{(b, q', m, \gamma, R) \in \delta(q, a, \alpha)} \chi(q, a, \alpha, b, q', m, \gamma, R),$$

where $\chi(q, a, \alpha, b, q', m, \gamma, R)$ is the sentence describing the transition in which, upon reading a in state q, if α evaluates to true in the τ-structure $\mathbf{A}$ currently held in the rs, then the machine writes b, enters state q', makes the move m, and replaces the relation $R^{\mathbf{A}}$ by $\gamma^{\mathbf{A}}$ in the rs. Assume that $m = L$ and S_j is the relation variable which encodes R, we write $\chi(q, a, \alpha, b, q', m, \gamma, R)$ as the conjunction of:

$$\forall \bar{p} \forall \bar{t} \Big(\neg (\forall \bar{x} (\bar{p} \preceq \bar{x})) \wedge T_a(\bar{p}, \bar{t}) \wedge H_q(\bar{p}, \bar{t}) \wedge \hat{\alpha}(\bar{t}) \rightarrow$$
$$\Big(T_b(\bar{p}, \bar{t} + 1) \wedge H_{q'}(\bar{p} - 1, \bar{t} + 1) \wedge$$
$$\forall \bar{x} \big(\neg (\bar{x} \sim \bar{p}) \rightarrow \big(\bigwedge_{i \in \{0, 1, \mathfrak{b}\}} T_i(\bar{x}, \bar{t} + 1) \leftrightarrow T_i(\bar{x}, \bar{t}) \big) \big) \wedge$$

$$\forall x_1 \ldots x_{r_j} \big(S_j(x_1, \ldots, x_{r_j}, \bar{t}+1) \leftrightarrow \hat{\gamma}(x_1, \ldots, x_{r_j}, \bar{t}) \wedge$$

$$\bigwedge_{0 \leq i \leq l, i \neq j} \big(\forall x_1 \ldots x_{r_i} (S_i(x_1, \ldots, x_{r_i}, \bar{t}) \leftrightarrow S_i(x_1, \ldots, x_{r_i}, \bar{t}+1))\big)\Big)$$

and

$$\forall \bar{p} \forall \bar{t} \Big(\forall \bar{x}(\bar{p} \preceq \bar{x}) \wedge T_a(\bar{p}, \bar{t}) \wedge H_q(\bar{p}, \bar{t}) \wedge \hat{\alpha}(\bar{t}) \rightarrow$$

$$\big(T_b(\bar{p}, \bar{t}+1) \wedge H_{q'}(\bar{p}, \bar{t}+1) \wedge$$

$$\forall \bar{x}\big(\neg(\bar{x} \sim \bar{p}) \rightarrow \big(\bigwedge_{i \in \{0,1,\mathfrak{b}\}} T_i(\bar{x}, \bar{t}+1) \leftrightarrow T_i(\bar{x}, \bar{t})\big)\big) \wedge$$

$$\forall x_1 \ldots x_{r_j} \big(S_j(x_1, \ldots, x_{r_j}, \bar{t}+1) \leftrightarrow \hat{\gamma}(x_1, \ldots, x_{r_j}, \bar{t}) \wedge$$

$$\bigwedge_{0 \leq i \leq l, i \neq j} \big(\forall x_1 \ldots x_{r_i} (S_i(x_1, \ldots, x_{r_i}, \bar{t}) \leftrightarrow S_i(x_1, \ldots, x_{r_i}, \bar{t}+1))\big)\Big)$$

where $\hat{\alpha}(\bar{t})$ and $\hat{\gamma}(x_1, \ldots, x_{r_j}, \bar{t})$ are the formulae obtained by replacing in the formula α and $\gamma(x_1, \ldots, x_{r_j})$, respectively, each atomic sub-formula of the form $R_i(y_1, \ldots, y_{r_i})$ $(0 \leq i \leq l)$ by $S_i(y_1, \ldots, y_{r_i}, \bar{t})$. We use abbreviations $\bar{p} - 1$ and $\bar{t} + 1$ for the predecessor of $\bar{p}$ and the successor of $\bar{t}$ in the lexicographic order induced by O, respectively; these are clearly definable in first-order logic. The second formula above is very similar to the first one, and handles the case when $\bar{p}$ is the left-most cell of the tape: then the head does not move and stays in p.

– Finally, a sentence stating that at some point, M is in an accepting final state.

$$\exists \bar{p} \exists \bar{t} (H_{q_m}(p, t)).$$

We show next that, M accepts a given σ-structure $\mathbf{I}$ iff there are relations closed under equivalence of FO^k-types of tuples as required by the SO^ω quantifiers in the prefix of φ_M, which assigned to the relation variables $O, T_0, T_1, T_\mathfrak{b}, H_{q_0}, \ldots, H_{q_m}, S_0, \ldots, S_l$ satisfy ψ.

Let $\mathcal{C}$ be the set of equivalence classes of k-tuples on $\mathbf{I}$ determined by the equivalence relation $\equiv^k$. Let $\leq^k$ be the partial order of Theorem 2 and let "$\leq^k$" be $\{([(a_{11}, \ldots, a_{1k})], [(a_{21}, \ldots, a_{2k})]) \in \mathcal{C}^2 : (a_{11}, \ldots, a_{1k}, a_{21}, \ldots, a_{2k}) \in \leq^k\}$. Since the relation "$\leq^k$" is a subset of $\mathcal{C}^2$, it follows from Fact 11 that there is a relation with the intended interpretation for O which is closed under $\equiv^{2k}$ on $\mathbf{I}$.

Regarding $T_0, T_1, T_\mathfrak{b}, H_{q_0}, \ldots, H_{q_m}$. If M accepts a given σ-structure $\mathbf{I}$, then for each $R_i \in \{T_0, T_1, T_\mathfrak{b}, H_{q_0}, \ldots, H_{q_m}\}$ there is a $\mathcal{C}_{R_i} \subseteq \mathcal{C}^{2 \cdot s}$ such that the corresponding relation $R_i^{\mathbf{I}} = \{(a_{11}, \ldots, a_{1k}, a_{21}, \ldots, a_{2k}, \ldots, a_{(2 \cdot s)1}, \ldots, a_{(2 \cdot s)k}) \in I^{2 \cdot s \cdot k} : ([(a_{11}, \ldots, a_{1k})], [(a_{21}, \ldots, a_{2k})], \ldots, [(a_{(2 \cdot s)1}, \ldots, a_{(2 \cdot s)k})]) \in \mathcal{C}_{R_i}\}$ meets the intended interpretation and, by Fact 11, is closed under $\equiv^{2 \cdot s \cdot k}$ on $\mathbf{I}$.

For each $R_i \in \tau$ of arity r_i, let $\mathcal{C}_i$ be the set of equivalence classes of r_i-tuples determined by $\equiv^k$ on $\mathbf{I}$. By Fact 12, in every configuration in a computation of a relational machine M of arity k on an input σ-structure $\mathbf{I}$, every relation R_i of arity r_i in its rs is closed under $\equiv^k$ on $\mathbf{I}$. Thus, if M accepts a given σ-structure $\mathbf{I}$, then for each relation variable $S_0, \ldots, S_l$ used to model the content of the relational store, there is a $\mathcal{C}_{S_i} \subseteq \mathcal{C}_i \times \mathcal{C}^s$ such that the corresponding relation

$S_i^{\mathbf{I}} = \{(a_1, \ldots, a_{r_i}, a_{11}, \ldots, a_{1k}, a_{21}, \ldots, a_{2k}, \ldots, a_{s1}, \ldots, a_{sk}) \in I^{r_i+s\cdot k} :$
$([(a_1, \ldots, a_{r_i})], [(a_{11}, \ldots, a_{1k})], [(a_{21}, \ldots, a_{2k})], \ldots, [(a_{s1}, \ldots, a_{sk})]) \in \mathcal{C}_{S_i}\}$,
meets the intended interpretation. Again by Fact 11, each of these relations $S_i^{\mathbf{I}}$ is closed under $\equiv^{(s+1)\cdot k}$ on $\mathbf{I}$.

We conclude that, for every $\mathbf{I} \in \mathcal{B}_\sigma$, $\mathbf{I} \models \varphi_M$ iff M accepts $\mathbf{I}$. □

5.1 SO$^\omega$ Captures the Relational Polynomial-Time Hierarchy

We show in this section the exact correspondence between the prenex fragments of SO$^\omega$ and the levels of the *relational* polynomial-time hierarchy.

To prove the following lemma, we adapt the strategy used for Turing machines in [9] to the case of relational machines.

Lemma 3. *If M is a nondeterministic relational machine with an oracle $\mathcal{A}$ in $\Sigma_m^{\mathrm{P}_r}$, for some $m \geq 0$, then there is a nondeterministic relational machine M' which is equivalent to M and which, in any computation, asks at most one query to an oracle $\mathcal{A}'$ which is also in $\Sigma_m^{\mathrm{P}_r}$.*

Proof. We assume that $M = \langle Q, \Sigma, \delta, q_0, \mathfrak{b}, F, \sigma, \tau, T, \Omega, \Phi\rangle$ works in nondeterministic time bounded by $(size_k(\mathbf{I}))^s$ for some $s \geq 1$ and input relational structure $\mathbf{I}$ of vocabulary $\sigma = \{E_1, \ldots, E_l\}$. We also assume that the subset of distinguished oracle relation symbols in the vocabulary τ of M is $\sigma^o = \{R_1^o, \ldots, R_n^o\}$, where for $1 \leq i \leq n$, the arity of R_i^o is r_i. We denote as $M_{\mathcal{A}}$ the relational machine in $\Sigma_m^{\mathrm{P}_r}$ which decides $\mathcal{A}$. We assume that the arity of $M_{\mathcal{A}}$ is k' and that it works in time bounded by $(size_{k'}(\mathbf{I}_o))^{s'}$ for some $s' \geq 1$ and input relational structure $\mathbf{I}_o$ of vocabulary σ^o.

M' works as follows. First, it guesses a sequence of oracle queries, i.e., a sequence of σ^o-structures, as well as their corresponding answers. M' does this by guessing and writing over its Turing machine tape a sequence of tuples of the form $((\bar{a}_1, \ldots, \bar{a}_n), A)$, where A is either Y or N and for $1 \leq i \leq n$, $\bar{a}_i \in \{0,1\}^{size_{r_i}(\mathbf{I})}$. Each n-tuple $(\bar{a}_1, \ldots, \bar{a}_n)$ represents a query to the oracle and A is the answer to that query. The σ^o-structure corresponding to a given n-tuples $(\bar{a}_1, \ldots, \bar{a}_n)$ is obtained by interpreting $R_i^o \in \sigma^o$ with the following corresponding relation

$$\{\bar{b} \in (dom(\mathbf{I}))^{r_i} : \bar{b} \text{ is in the } j\text{-th equivalence class in the order given by } \leq^{r_i} \text{ and the } j\text{-th component of } \bar{a}_i \text{ equals } 1\}.$$

Note that, as shown in the proof of Proposition 3, given a tuple $\bar{a}_i \in \{0,1\}^{size_{r_i}(\mathbf{I})}$, M' can compute the corresponding relation and store it in its rs working in time bounded by a polynomial in $size_{r_i}(\mathbf{I})$.

Then, M' proceeds as M, except that every time that M makes a query to the oracle, M' just takes the answer guessed for that query at the beginning of the computation.

Let us fix an arbitrary computation of M' on the input structure $\mathbf{I}$. At the end of the computation, M' must check that the guessed queries match the sequence of real queries of M in the fixed computation, and that the guessed answers for those queries coincide with the actual answers from the oracle $\mathcal{A}$. If any of those conditions is not met, M' stops in a rejecting state.

To check whether the guessed yes answers are correct, M' simply takes each guessed tuple of the form $((\bar{a}_1, \ldots, \bar{a}_n), Y)$, stores their corresponding relations in its rs, and works as $M_{\mathcal{A}}$ to check whether the structure of domain $dom(\mathbf{I})$ formed by those relations, is in the oracle $\mathcal{A}$. M' does not need to call an oracle to do this since the answer guessed for those queries is yes.

As to the queries for which the guessed answer is no, M' does need to use an oracle, but it suffices to make just one query to it. To do this, M' encodes the guessed oracle queries with guessed answers no as a structure of vocabulary $\sigma^N = \{E_1^o, \ldots, E_l^o, S_1^o, \ldots, S_n^o\}$, where for $1 \leq i \leq j$, $arity(E_i^o) = arity(E_i)$, and for $1 \leq i \leq n$, $arity(S_i^o) = r_i + s \cdot k$. Actually, σ^N is the set of distinguished oracle relation symbols in the vocabulary τ' of the rs of M'.

Let $(\bar{a}_{11}, \ldots, \bar{a}_{1n}), \ldots, (\bar{a}_{m1}, \ldots, \bar{a}_{mn})$ be the sequence of tuples corresponding to the queries with guessed negative answers, for each $1 \leq u \leq n$, M' stores in its rs the following relation:

$$S_u^o := \bigcup_{1 \leq i \leq m} \{(\bar{b}, \bar{t}) \in (dom(\mathbf{I}))^{r_u + s \cdot k} : \bar{b} \text{ is in the } j\text{-th equivalence class in the order given by } \leq^{r_u}, \text{ the } j\text{-th component of } \bar{a}_{iu} \text{ equals 1, } \bar{t} \text{ and belongs to the } i\text{-th equivalence class in the order given by } \leq^{s \cdot k}\}.$$

Furthermore, for $1 \leq i \leq l$, M' stores in E_i^o in its rs the relation $E_i^{\mathbf{I}}$ in the input structure $\mathbf{I}$. Then it asks the oracle $\mathcal{A}'$ the query represented by the σ^N-structure of domain $dom(\mathbf{I})$ formed by those relations. If the answer of the oracle $\mathcal{A}'$ is yes, then the no answers guessed by M' are all correct.

Let $\mathbf{J} \in \mathcal{B}_{\sigma^N}$, let $\mathbf{J}_\sigma$ be the structure $\mathbf{J}$ restricted to the vocabulary $\{E_1^o, \ldots, E_l^o\}$, and let $\mathcal{C}$ be the set of equivalence classes of k-tuples determined by the equivalence relation $\equiv^k$ on $\mathbf{J}_\sigma$. For $([\bar{t}_1], \ldots, [\bar{t}_s]) \in \mathcal{C}^s$ and $1 \leq i \leq n$, we denote as $T^{\mathbf{J}}_{i,([\bar{t}_1],\ldots,[\bar{t}_s])}$ the following relation:

$$\{(a_1, \ldots, a_{r_i}) \in (dom(\mathbf{J}))^{r_i} : \text{for } \bar{t}'_1 \in [\bar{t}_1], \ldots, \bar{t}'_s \in [\bar{t}_s], \text{ it holds that } \mathbf{J} \models S_i^o(a_1, \ldots, a_{o_i}, \bar{t}'_1, \ldots, \bar{t}'_s)\}$$

Accordingly, we denote as $\mathbf{J}_{([\bar{t}_1],\ldots,[\bar{t}_s])}$ the structure of vocabulary σ^o and domain $dom(\mathbf{J})$ which is obtained by interpreting the relation symbols $R_1^o, \ldots, R_n^o$ with the relations $T^{\mathbf{J}}_{1,([\bar{t}_1],\ldots,[\bar{t}_s])}, \ldots T^{\mathbf{J}}_{n,([\bar{t}_1],\ldots,[\bar{t}_s])}$, respectively.

A given structure $\mathbf{J} \in \mathcal{B}_{\sigma^N}$ belongs to the new oracle $\mathcal{A}'$ iff, for every s-tuple $([\bar{t}_1], \ldots, [\bar{t}_s])$ of equivalence classes in $\mathcal{C}^s$, it holds that $\mathbf{J}_{([\bar{t}_1],\ldots,[\bar{t}_s])} \notin \mathcal{A}$.

The Machine $M_{\mathcal{A}'}$ which decides the relational language $\mathcal{A}'$ works as follows:

1. $M_{\mathcal{A}'}$ computes the preorder $\leq^k$ of Theorem 2 on $\mathbf{J}_\sigma$.
2. Using $\leq^k$, $M_{\mathcal{A}'}$ computes $size_k(\mathbf{J}_\sigma)$.
3. Let $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_s)$ and $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_s)$ be $(k \cdot s)$-tuples, we say that $\bar{x} \preceq \bar{y}$ if $([\bar{x}_1], \ldots, [\bar{x}_s])$ precedes or equals $([\bar{y}_1], \ldots, [\bar{y}_s])$ in the lexicographic order induced by $\leq^k$ on $\mathcal{C}^s$. If $\bar{x} \preceq \bar{y}$ and $\bar{y} \preceq \bar{x}$, then we say that $\bar{x} \sim \bar{y}$.
   $X := \forall \bar{y}(\bar{x} \preceq \bar{y})$;
   $Y := \neg \bar{x} = \bar{x}$;
   While $\exists \bar{x}(\neg Y(\bar{x}))$;

Begin
$R_1 := \exists \bar{x}(X(\bar{x}) \wedge S_1^o(y_1, \ldots, y_{r_1}, \bar{x}))$;
$\vdots$
$R_n := \exists \bar{x}(X(\bar{x}) \wedge S_n^o(y_1, \ldots, y_{r_n}, \bar{x}))$;
$M_{\mathcal{A}'}$ works as $M_{\mathcal{A}}$ taking as input the σ^o-structure of domain $dom(\mathbf{J})$ formed by the relations $R_1, \ldots, R_n$ held in its rs;
If "$M_{\mathcal{A}}$ accepts" then
    $M_{\mathcal{A}'}$ stops in a rejecting state;
$Y := Y(\bar{x}) \vee X(\bar{x})$;
$X := \neg Y(\bar{x}) \wedge$
    $\forall \bar{y}\big(\big(Y(\bar{y}) \wedge \forall \bar{z}(Y(\bar{z}) \rightarrow \bar{z} \preceq \bar{y})\big) \rightarrow$
        $(\bar{y} \preceq \bar{x} \wedge \forall \bar{z}((\bar{y} \preceq \bar{z} \wedge \bar{z} \preceq \bar{x}) \rightarrow (\bar{y} \sim \bar{z} \vee \bar{z} \sim \bar{x})))\big)$;
End;
$M_{\mathcal{A}'}$ stops in an accepting state.

Since the "while loop" in the previous algorithm is executed $(size_k(\mathbf{J}_\sigma))^s$ times, it is not difficult to see that $M_{\mathcal{A}'}$ works in nondeterministic time bounded by a polynomial in $size_{k'}(\mathbf{J})$, where $k' \geq s \cdot k$ is the arity of $M_{\mathcal{A}'}$. □

Theorem 13. *For $m \geq 1$, $\Sigma_m^{1,\omega}$ captures $\Sigma_m^{\mathrm{P}_r}$.*

Proof. **a) $\Longrightarrow$:** First, we show that for every relational vocabulary σ, every $\Sigma_m^{1,\omega}[\sigma]$-sentence φ can be evaluated in $\Sigma_m^{\mathrm{P}_r}$.

Suppose that φ is $\exists^{k_{11}} X_{11} \ldots \exists^{k_{1s_1}} X_{1s_1} \forall^{k_{21}} X_{21} \ldots \forall^{k_{2s_2}} X_{2s_2} \exists^{k_{31}} X_{31} \ldots \exists^{k_{3s_3}} X_{3s_3} \ldots Q^{k_{m1}} X_{m1} \ldots Q^{k_{ms_m}} X_{ms_m}(\psi)$, where Q is either $\exists$ or $\forall$, depending on whether m is odd or even, respectively, and ψ is a first-order formula of vocabulary $\sigma \cup \{X_{11}, \ldots, X_{1s_1}, X_{21}, \ldots, X_{2s_2}, \ldots, X_{m1}, \ldots, X_{ms_m}\}$.

We build a nondeterministic relational oracle machine M_φ which evaluates φ on input structures of vocabulary σ. For $1 \leq j \leq s_1$, let $k'_{1j} \geq 2k_{1j}$ be the arity of the relational machine $M_{\leq^{k_{1j}}}$ of Lemma 2 which computes the preorder $\leq^{k_{1j}}$ of Theorem 2. The arity k of M_φ is $max(\{k'_{11}, \ldots, k'_{1s_1}\})$. The vocabulary τ of the relational store is $\sigma \cup \sigma^{o_{m-1}} \cup \{\leq^{k_{11}}, \ldots, \leq^{k_{1s_1}}, S_1, \ldots, S_{s_1}\}$, where for $1 \leq j \leq s_1$, the arity of $\leq^{k_{1j}}$ is $2k_{1j}$ and the arity of S_j is k_{1j}, and $\sigma^{o_{m-1}} = \{R^{o_{m-1}} : R \in \sigma\} \cup \{X_{11}^{o_{m-1}}, \ldots, X_{1s_1}^{o_{m-1}}\}$ is the set of distinguished oracle relation symbols. For every $R \in \sigma$, the arity of $R^{o_{m-1}}$ is the same as the arity of R, and for $1 \leq j \leq s_1$, the arity of $X_{1j}^{o_{m-1}}$ is the same as the arity $r_{1j} \leq k_{1j}$ of X_{1j}.

Let φ'_{m-1} be the following sentence:

$$\exists^{k_{21}} X_{21} \ldots \exists^{k_{2s_2}} X_{2s_2} \neg (\exists^{k_{31}} X_{31} \ldots \exists^{k_{3s_3}} X_{3s_3} \ldots Q^{k_{m1}} X_{m1} \ldots Q^{k_{ms_m}} X_{ms_m}(\psi'_{m-1})),$$

where ψ'_{m-1} is ψ with every occurrence of a relation symbol $R \in \sigma$ replaced by the corresponding relation symbol $R^{o_{m-1}} \in \sigma^{o_{m-1}}$, and every occurrence of a relation variable X_{1j} $(1 \leq j \leq s_1)$ replaced by the corresponding relation symbol $X_{1j}^{o_{m-1}} \in \sigma^{o_{m-1}}$. The oracle $\mathcal{C}_{m-1}$ of M_φ is the relational language $\{\mathbf{A} \in \mathcal{B}_{\sigma^{o_{m-1}}} : \mathbf{A} \models \varphi'_{m-1}\}$.

On an input structure $\mathbf{I}$, M_φ works as follows:

1. For every $1 \leq j \leq s_1$, M_φ builds the preorder $\leq^{k_{1j}}$ of Theorem 2 in its rs.
2. M_φ computes $size_{k_{1j}}(\mathbf{I})$ for every $1 \leq j \leq s_1$.
3. For every $1 \leq j \leq s_1$, M_φ guesses and writes over its Turing machine tape a tuple $\bar{a}_j \in \{0,1\}^{size_{k_{1j}}(\mathbf{I})}$.
4. Using the binary tuples guessed in the previous step, M_φ generates, for every $1 \leq j \leq s_1$, a relation which is placed in the distinguish oracle relation X^o_{1j} of its rs and is closed under the equivalence relation $\equiv^{k_{1j}}$ in $\mathbf{I}$. M_φ works by storing in X^o_{1j} the tuples in all l-th equivalence classes in the order given by $\leq^{k_{1j}}$ for which the l-th component of $\bar{a}_j$ equals 1.
5. Finally, for every $R \in \sigma$, M_φ stores the relation $R^{\mathbf{I}}$ into the corresponding oracle relation $R^{o_{m-1}}$ and moves to the oracle query state $q_?$. M_φ accepts the input structure $\mathbf{I}$ iff the relational structure of domain I formed by the distinguished set of oracles relations currently held in its rs does not belongs to the oracle set $\mathcal{C}_{m-1}$, i.e., iff M_φ transfers from the state $q_?$ into the state q_{NO}.

As shown in the proof of Proposition 3, M_φ can perform tasks 1 to 4 working in time bounded by a polynomial in $size_k(\mathbf{I})$. Furthermore, task 5 can clearly be performed in constant time by M_φ.

Therefore, it only remains to show that the oracle $\mathcal{C}_{m-1}$ is in $\Sigma^{\mathrm{P}_r}_{m-1}$, i.e., that there is a nondeterministic relational machine $M_{\varphi'_{m-1}}$ such that $\mathcal{L}(M_{\varphi'_{m-1}}) = \mathcal{C}_{m-1}$ and $\mathcal{L}(M_{\varphi'_{m-1}}) \in \Sigma^{\mathrm{P}_r}_{m-1}$.

$M_{\varphi'_{m-1}}$ evaluates φ'_{m-1} on input structures of vocabulary $\sigma^{o_{m-1}}$. The vocabulary τ'_1 of the relational store is $\sigma^{o_{m-1}} \cup \sigma^{o_{m-2}} \cup \{\leq^{k_{21}}, \ldots, \leq^{k_{2s_2}}, S_1, \ldots, S_{s_2}\}$, where for $1 \leq j \leq s_2$, the arity of $\leq^{k_{2j}}$ is $2k_{2j}$ and the arity of S_j is k_{2j}, and $\sigma^{o_{m-2}} = \{R^{o_{m-2}} : R \in \sigma^{o_{m-1}}\} \cup \{X^{o_{m-2}}_{21}, \ldots, X^{o_{m-2}}_{2s_2}\}$ is the set of distinguished oracle relation symbols. For every $R \in \sigma^{o_{m-1}}$, the arity of $R^{o_{m-2}}$ is the same as the arity of R, and for $1 \leq j \leq s_2$, the arity of $X^{o_{m-2}}_{2j}$ is the same as the arity $r_{2j} \leq k_{2j}$ of X_{2j}.

The oracle $\mathcal{C}_{m-2}$ of $M_{\varphi'_{m-1}}$ is the relational language $\{\mathbf{A} \in \mathcal{B}_{\sigma^{o_{m-2}}} : \mathbf{A} \models \varphi'_{m-2}\}$, where φ'_{m-2} is $\exists^{k^3_1} X_{31} \ldots \exists^{k^3_{s_3}} X_{3s_3} \ldots Q^{k^m_1} X_{m1} \ldots Q^{k^m_{s_m}} X_{ms_m}(\psi'_{m-2})$. Here ψ'_{m-2} is ψ'_{m-1} with every occurrence of a relation symbol $R \in \sigma^{o_{m-1}}$ replaced by the corresponding relation symbol $R^{o_{m-2}} \in \sigma^{o_{m-2}}$, and every occurrence of a relation variable X_{2j} $(1 \leq j \leq s_2)$ replaced by the corresponding relation symbol $X^{o_{m-2}}_{2j} \in \sigma^{o_{m-2}}$.

The way in which the machine $M_{\varphi'_{m-1}}$ works is exactly the same as the way in which the original machine M_φ works, i.e., $M_{\varphi'_{m-1}}$ executes steps 1 to 5 adapted to the vocabulary τ'_1 of its rs and for $1 \leq j \leq s_2$. Therefore, for every input structure $\mathbf{I}'$ of vocabulary $\sigma^{o_{m-1}}$, $M_{\varphi'_{m-1}}$ works in nondeterministic relational time bounded by a polynomial in $size_{k'}(\mathbf{I}')$, where $k' = max(\{k'_{21}, \ldots, k'_{2s_2}\}))$ is the arity of $M_{\varphi'_{m-1}}$, and for $1 \leq j \leq s_2$, $k'_{2j} \geq 2k_{2j}$ is

the arity of the relational machine $M_{\leq^{k_{2j}}}$ of Lemma 2 which computes the preorder $\leq^{k_{2j}}$ of Theorem 2.

This process continues in the same way for the blocks 3 to $m-1$ of quantifiers in φ. Since for the last block m of quantifiers the resulting φ'_1 is either

$$\exists^{k_1^m} X_{m1} \ldots \exists^{k_{s_m}^m} X_{ms_m}(\psi'_1) \qquad \text{or} \qquad \exists^{k_1^m} X_{m1} \ldots \exists^{k_{s_m}^m} X_{ms_m}(\neg\psi'_1),$$

it follows by Theorem 7 that the oracle $\mathcal{C}_1$ of $M_{\varphi'_2}$ (i.e., the relational language $\{\mathbf{A} \in \mathcal{B}_{\sigma^{o_1}} : \mathbf{A} \models \varphi'_1\}$) is in $\mathrm{NP}_r = \Sigma_1^{\mathrm{P}_r}$.

Hence, φ can be evaluated in $\Sigma_m^{\mathrm{P}_r}$.

b) $\Longleftarrow$: Next, we show that every $\Sigma_m^{\mathrm{P}_r}$ property of finite relational structures can be expressed in $\Sigma_m^{1,\omega}$.

We use induction on m. The base case is Proposition 4. Now consider a Boolean query $q : \mathcal{B}_\sigma \to \{0,1\}$ in $\Sigma_m^{\mathrm{P}_r}$ where $m > 1$. Let M be the nondeterministic relational machine with an oracle in $\Sigma_{m-1}^{\mathrm{P}_r}$ which computes q. Let $\{\mathbf{I} \in \mathcal{B}_{\sigma^o} : q_o(\mathbf{I}) = 1\}$, where σ^o is the set of distinguished oracle relation symbols of M and q_o is a boolean query in $\Sigma_{m-1}^{\mathrm{P}_r}$, be the oracle of M.

By inductive hypothesis, for every boolean query q_i in $\Sigma_{m-1}^{P_r}$, there is a sentence $\alpha_{q_i} \in \Sigma_{m-1}^{1,\omega}$ which express q_i. In particular, there is a sentence $\alpha_{q_o} \in \Sigma_{m-1}^{1,\omega}$ of vocabulary σ^o, which express the boolean query q_o. Let α_{q_o} be σ^o-sentence $\exists^{k_{21}} X_{21} \ldots \exists^{k_{2s_2}} X_{2s_2} \forall^{k_{31}} X_{31} \ldots \forall^{k_{3s_3}} X_{3s_3} \ldots Q^{k_{m1}} X_{m1} \ldots Q^{k_{ms_m}} X_{ms_m}(\psi_o)$, where Q is either $\exists$ or $\forall$, depending on whether m is odd or even, respectively, and ψ_o is a first-order sentence of vocabulary $\sigma^o \cup \{X_{21}, \ldots, X_{2s_2}, X_{31}, \ldots, X_{3s_3}, \ldots, X_{m1}, \ldots, X_{ms_m}\}$.

We show how to modify the formula φ_M in Proposition 4, i.e., the formula corresponding to the nondeterministic relational machine, to reflect the interaction of M with its oracle. We assume that M works in time $(size_k(\mathbf{I}))^s$ for some $s \geq 1$ and $\mathbf{I} \in \mathcal{B}_\sigma$.

First, we add to the prefix of φ_M the existential quantification $\exists^{(s+1)\cdot k} S_1^o \ldots \exists^{(s+1)\cdot k} S_n^o$. Let r_i^o denote the arity of the distinguished oracle relation $R_i^o \in \sigma^o = \{R_1^o, \ldots, R_n^o\}$. For $1 \leq i \leq n$, the arity of the relation variable S_i^o is $r_i^o + s \cdot k$. The intended interpretation of $S_i^o(\bar{a}, \bar{t})$ is that at time $\bar{t}$, the distinguished oracle relation R_i^o in the rs contains the r_i^o-tuple $\bar{a}$.

The sub-formula ψ of φ_M treats the variables $S_1^o, \ldots, S_n^o$ corresponding to the distinguished oracle relations in the rs of M in exactly the same way as the variables $S_1, \ldots, S_l$ which correspond to the other relations in the rs of M. We only need to add a special case to the sub-formula of ψ which express that the relations T_i's, H_q's, S_i's and S_i^o's respect the transition function of M. When M is in the oracle query state $q_?$, $\chi(q_?, a, \alpha, b, q', m, \gamma, R)$ is the sentence describing the transition in which upon entering the query state $q_?$, the machine moves to state q_{YES} if the σ^o-structure held in the rs is in the oracle of M, or to state q_{NO} if it is not. W.l.o.g., we assume that the contents of the rs as well as of the working tape of M and the position of its read/write head remain unchanged.

The more "natural" way of expressing $\chi(q_?, a, \alpha, b, q', m, \gamma, R)$ is probably as follows:

$$\forall\bar{p}\forall\bar{t}\Big(H_{q_?}(\bar{p},\bar{t}) \rightarrow \big(\hat{\alpha}_{q_o}(\bar{t}) \rightarrow H_{q_{YES}}(\bar{p},\bar{t}+1)\big) \wedge \big(\neg\hat{\alpha}_{q_o}(\bar{t}) \rightarrow H_{q_{NO}}(\bar{p},\bar{t}+1)\big) \wedge$$
$$\forall\bar{x}\big(\textstyle\bigwedge_{i\in\{0,1,\mathfrak{b}\}} T_i(\bar{x},\bar{t}+1) \leftrightarrow T_i(\bar{x},\bar{t})\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq l} \big(\forall x_1 \ldots x_{r_i}(S_i(x_1,\ldots,x_{r_i},\bar{t}) \leftrightarrow S_i(x_1,\ldots,x_{r_i},\bar{t}+1))\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq n} \big(\forall x_1 \ldots x_{r_i^o}(S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}) \leftrightarrow S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}+1))\big)\Big)$$

where $\hat{\alpha}_{q_o}(\bar{t})$ is the formula obtained by replacing in α_{q_o} each atomic subformula of the form $R_i^o(y_1,\ldots,y_{r_i^o})$ $(0 \leq i \leq n)$ by $S_i^o(y_1,\ldots,y_{r_i^o},\bar{t})$. But, we need the resulting formula to be in prenex normal form. Unfortunately, equivalences such as $\forall x\, Q\, \gamma(x) \leftrightarrow \forall X\, Q\, \big(\exists! x X(x) \rightarrow \forall x(X(x) \rightarrow \gamma(x))\big)$, where Q stands for an arbitrary sequence of first- and second-order quantifiers and $\exists! x X(x)$ means "there exists exactly one x such that $X(x)$", are no longer true for SO^ω. Not all elements of the domain are distinguishable from each other in FO^k for a fixed k. Thus, it may well happen that there is an element a in the domain of a given structure $\mathbf{I}$ such that $\{a\}$ is not closed under under $\equiv^k$ on $\mathbf{I}$.

In order to write $\chi(q_?, a, \alpha, b, q', m, \gamma, R)$ in a form such that the SO^ω quantifiers in α_{q_o} can be moved to the prefix of φ_M, we use Lemma 3 . That is, we assume that in any computation M makes at most one query to its oracle. Under this assumption, we can then write $\chi(q_?, a, \alpha, b, q', m, \gamma, R)$ as the conjunction of:

$$\exists^{k_{21}} X_{21} \ldots \exists^{k_{2s_2}} X_{2s_2} \forall^{k_{31}} X_{31} \ldots \forall^{k_{3s_3}} X_{3s_3} \ldots Q^{k_{m1}} X_{m1} \ldots Q^{k_{ms_m}} X_{ms_m}$$
$$\forall\bar{p}\forall\bar{t}\Big(H_{q_?}(\bar{p},\bar{t}) \wedge \hat{\psi}_o(\bar{t}) \rightarrow$$
$$H_{q_{YES}}(\bar{p},\bar{t}+1) \wedge \forall\bar{x}\big(\textstyle\bigwedge_{i\in\{0,1,\mathfrak{b}\}} T_i(\bar{x},\bar{t}+1) \leftrightarrow T_i(\bar{x},\bar{t})\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq l} \big(\forall x_1 \ldots x_{r_i}(S_i(x_1,\ldots,x_{r_i},\bar{t}) \leftrightarrow S_i(x_1,\ldots,x_{r_i},\bar{t}+1))\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq n} \big(\forall x_1 \ldots x_{r_i^o}(S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}) \leftrightarrow S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}+1))\big)\Big)$$

and

$$\forall^{k_{21}} X'_{21} \ldots \forall^{k_{2s_2}} X'_{2s_2} \exists^{k_{31}} X'_{31} \ldots \exists^{k_{3s_3}} X'_{3s_3} \ldots Q^{k_{m1}} X'_{m1} \ldots Q^{k_{ms_m}} X'_{ms_m}$$
$$\forall\bar{p}\forall\bar{t}\Big(H_{q_?}(\bar{p},\bar{t}) \wedge \neg\hat{\psi}'_o(\bar{t}) \rightarrow$$
$$H_{q_{NO}}(\bar{p},\bar{t}+1) \wedge \forall\bar{x}\big(\textstyle\bigwedge_{i\in\{0,1,\mathfrak{b}\}} T_i(\bar{x},\bar{t}+1) \leftrightarrow T_i(\bar{x},\bar{t})\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq l} \big(\forall x_1 \ldots x_{r_i}(S_i(x_1,\ldots,x_{r_i},\bar{t}) \leftrightarrow S_i(x_1,\ldots,x_{r_i},\bar{t}+1))\big) \wedge$$
$$\textstyle\bigwedge_{0\leq i\leq n} \big(\forall x_1 \ldots x_{r_i^o}(S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}) \leftrightarrow S_i^o(x_1,\ldots,x_{r_i^o},\bar{t}+1))\big)\Big)$$

where $\hat{\psi}_o(\bar{t})$ is the formula obtained by replacing in ψ_o each atomic sub-formula of the form $R_i^o(y_1, \ldots, y_{r_i^o})$ $(0 \leq i \leq n)$ by $S_i^o(y_1, \ldots, y_{r_i^o}, \bar{t})$, and $\hat{\psi}'_o(\bar{t})$ is the formula obtained by replacing in $\hat{\psi}_o$ each occurrence of a relation variable $X_{ij} \in \{X_{21}, \ldots, X_{2s_2}, X_{31}, \ldots, X_{3s_3}, \ldots, X_{m1}, \ldots, X_{ms_m}\}$ by X'_{ij}. Note that for the sentence above, we use the fact that $\neg\alpha_{q_o}$ is equivalent to

$$\forall^{k_{21}} X_{21} \ldots \forall^{k_{2s_2}} X_{2s_2} \exists^{k_{31}} X_{31} \ldots \exists^{k_{3s_3}} X_{3s_3} \ldots Q^{k_{m1}} X_{m1} \ldots Q^{k_{ms_m}} X_{ms_m}(\neg\psi_o).$$

It is not difficult to see that the SO^ω quantifiers in the sentence above, can now be safely moved to the prefix of ψ_M and rearranged in such a way that the resulting formula is in $\Sigma_m^{1,\omega}$. □

Acknowledgements. We are deeply grateful to Lauri Hella for the interesting and stimulating discussions we had on this subject.

References

1. Chandra, A.K., Harel, D.: Computable queries for relational data bases. J. Comput. Syst. Sci. 21, 156–178 (1980)
2. Hull, R., Su, J.: Untyped sets, invention, and computable queries. In: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania, pp. 347–359. ACM Press, New York (1989)
3. Abiteboul, S., Vianu, V.: Generic computation and its complexity. In: Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing, pp. 209–219. ACM Press, New York (1991)
4. Abiteboul, S., Vianu, V.: Computing with first-order logic. J. Comput. Syst. Sci. 50, 309–335 (1995)
5. Abiteboul, S., Papadimitriou, C.H., Vianu, V.: Reflective relational machines. Inf. Comput. 143, 110–136 (1998)
6. Abiteboul, S., Vardi, M.Y., Vianu, V.: Fixpoint logics, relational machines, and computational complexity. J. ACM 44, 30–56 (1997)
7. Abiteboul, S., Vardi, M.Y., Vianu, V.: Computing with infinitary logic. Theor. Comput. Sci. 149, 101–128 (1995)
8. Dawar, A.: A restricted second order logic for finite structures. Inf. Comput. 143, 154–174 (1998)
9. Stockmeyer, L.J.: The polynomial-time hierarchy. Theor. Comput. Sci. 3, 1–22 (1976)
10. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Perspectives in Mathematical Logic. Springer, Heidelberg (1999)
11. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Redwood City (1994)
12. Libkin, L.: Elements Of Finite Model Theory. Texts in Theoretical Computer Science, EATCS. Springer, Heidelberg (2004)
13. Kolaitis, P.G., Vardi, M.Y.: Fixpoint logic vs. infinitary logic in finite-model theory. In: Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California, USA, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1992)

14. Kolaitis, P.G., Vardi, M.Y.: Infinitary logics and 0-1 laws. Inform. and Comput. 98, 258–294 (1992)
15. Dawar, A.: Feasible Computation Through Model Theory. PhD thesis, University of Pennsylvania, Philadelphia (1993)
16. Dawar, A., Lindell, S., Weinstein, S.: Infinitary logic and inductive definability over finite structures. Inf. Comput. 119, 160–175 (1995)
17. Otto, M.: Bounded variable logics and counting – A study in finite models, vol. 9. Springer, Heidelberg (1997)
18. Turull Torres, J.M.: A study of homogeneity in relational databases. Ann. Math. Artif. Intell. 33, 379–414 (2001), Also see erratum in 42(4), 443–444 (2004)

Qualitative Knowledge Discovery*

Gabriele Kern-Isberner[1], Matthias Thimm[1], and Marc Finthammer[2]

[1] Faculty of Computer Science, Technische Universität Dortmund, Germany
[2] Department of Computer Science, FernUniversität in Hagen, Germany

Abstract. Knowledge discovery and data mining deal with the task of finding useful information and especially rules in unstructured data. Most knowledge discovery approaches associate conditional probabilities to discovered rules in order to specify their strength. In this paper, we propose a qualitative approach to knowledge discovery. We do so by abstracting from actual probabilities to qualitative information and in particular, by developing a method for the computation of an ordinal conditional function from a possibly noisy probability distribution. The link between structural and numerical knowledge is established by a powerful algebraic theory of conditionals. By applying this theory, we develop an algorithm that computes sets of default rules from the qualitative abstraction of the input distribution. In particular, we show how sparse information can be dealt with appropriately in our framework. By making use of the duality between inductive reasoning and knowledge discovery within the algebraic theory of conditionals, we can ensure that the discovered rules can be considered as being most informative in a strict, formal sense.

1 Introduction

Knowledge discovery is the overall process to extract new and useful information from statistical data, with a focus on finding patterns and relationships that reveal generic knowledge, i. e., knowledge that is not specific to a certain situation. Moreover, these relationships should be presented to the user in an intelligible manner. This makes rules appropriate candidates to encode knowledge that is searched for, as they establish (often generic) relationships between isolated facts and are easily comprehensible for human beings. Usually, a conditional probability is associated with each rule by the knowledge discovery process to specify the strength, or the confidence of the rule.

However, while probabilities are a really expressive means to represent knowledge, they are often of only limited use when it comes to commonsense reasoning. First, there is no straightforward way to process probabilistic information. For instance, if the rules "If symptom A then disease D with probability 0.632" and "If symptom B then disease D with probability 0.715" are shown to the user, what should he believe if the patient he is facing has symptoms A *and* B? Second, while probabilities are appreciated for their (seemingly objective) preciseness, users would not feel comfortable if they had to distinguish sharply between, say, 0.715 and 0.721. Moreover, statistical data are often noisy and may show particularities of the population they were taken from, which

* The research reported here was partially supported by the Deutsche Forschungsgemeinschaft (grants BE 1700/7-1 and KE 1413/2-1).

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 77–102, 2008.

does not match the aim of discovering generic, context-independent knowledge. This suggests that precise probabilistic information is neither completely satisfactory nor useful for knowledge discovery.

In this paper, we propose to solve such problems by extracting more coarse-grained rules from data which are only equipped with an order of magnitude of the corresponding probability. Such qualitative rules could be used to reveal plausible relationships to the user, or even as default rules for commonsense reasoning, by applying one of the well-known nonmonotonic inference formalisms (cf. e.g. [1,2,3]). This perspective of discovering rules from data and feeding them into an inference engine to make inductive reasoning possible will play a decisive part for the methodology to be presented in this paper. More precisely, we will consider knowledge discovery and inductive reasoning as reverse processes (illustrated in Figure 1) – knowledge discovery extracts most relevant partial knowledge that may serve as a basis for further reasoning from frequency distributions representing complete probabilistic information, while inductive model-based reasoning builds up a complete epistemic model from partial knowledge in a knowledge base.

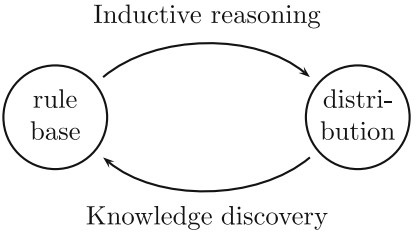


Fig. 1. Knowledge discovery and inductive reasoning as reverse processes

We build upon previous work. In [4,5], these ideas have been developed and implemented in a fully probabilistic framework. But the core methodology used in these papers is based on structural, algebraic techniques for abstract conditionals and can also be applied in a qualitative framework. However, we first have to transform probabilistic information obtained from data to qualitative rankings. For this, we modify the well-known approach for infinitesimal probabilities [6,1] to obtain a so-called *ordinal conditional function* [7] which assigns qualitative degrees of disbelief, or rankings, respectively, to propositions and conditionals. The level of qualitative abstraction of probabilities is determined by a parameter ε that specifies a measure of similarity between probabilistic values, according to the needs of the user.

Our approach offers a couple of nice advantages. First, the same methodology is used both for learning and reasoning, handling structural knowledge in a profound algebraic way. Second, the notion of relevance which is crucial for knowledge discovery can be given a precise meaning – rules are relevant wrt. a given set of (already discovered) rules, if they provide additional information for the inductively built model. Third, the qualitative information derived from data reflects an intuitive similarity of the probabilities, different from the approach in [8] in which sums of probabilities have to be used.

The outline of the paper is as follows. In the next section, we will recall basic facts on probabilistic reasoning and ordinal conditional functions. In section 3, we present our approach to extract qualitative information from statistical data. We also propose a heuristic how to find a proper abstraction parameter ε. Section 4 describes the core methodology which can be used for inductive representation and knowledge discovery and that is applied in section 5 for the knowledge discovery task. Based on this theoretical work, an algorithm for discovering default rules in statistical data is represented in section 6. Section 7 concludes the paper with a summary and an outlook on further work.

2 Inductive Reasoning with Probabilities and Rankings

We consider a propositional framework over a finite set $\mathcal{V} = \{V_1, V_2, \ldots\}$ of (multi-valued) propositional variables V_i with finite domains. For each variable $V_i \in \mathcal{V}$, the values are denoted by v_i. In generalizing the bivalued propositional framework, we call expressions of the form $V_i = v_i$ *literals*, and abbreviate them by v_i. The language $\mathcal{L}$ consists of all formulas A built by conjoining finitely many literals by conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) in a well-formed way. The conjunction operator, $\wedge$, will usually be omitted, so AB will mean $A \wedge B$, and negation is indicated by overlining, i. e., $\overline{A} = \neg A$. An *elementary conjunction* is a conjunction consisting of literals, and a *complete conjunction* is an elementary conjunction where each variable from $\mathcal{V}$ is instantiated by exactly one value. Let Ω denote the set of complete conjunctions of $\mathcal{L}$. Ω can be taken as the set of *possible worlds* ω, providing a complete description of each possible state, and hence corresponding to elementary events in probability theory.

Conditionals are written in the form $(B|A)$, with antecedents, A, and consequents, B, both formulas in $\mathcal{L}$, and may be read as uncertain rules of the form *if* A *then* B. Let $(\mathcal{L}|\mathcal{L})$ denote the set of all conditionals over $\mathcal{L}$. *Single-elementary conditionals* are conditionals whose antecedents are elementary conjunctions, and whose consequents consist of one single literal. To provide semantics for conditionals, a richer epistemic framework is needed than a plain bivalued semantics. Basically, for a conditional $(B|A)$ to be accepted, its confirmation, AB, must be more probable, plausible etc. than its refutation, $A\overline{B}$. Moreover, numerical degrees of probability, plausibility and the like can be assigned to conditionals to specify the strength with which they are believed, according to the chosen epistemic framework. In this paper, we will use probabilities to model a fully quantitative frame, and so-called *ordinal conditional functions,* OCFs, (or simply *ranking functions*) to model a qualitative, respectively semi-quantitative frame. We will briefly summarize basic facts on both modelling frames in the following. We will also address the problem which is crucial to this paper: Given partial information in form of a conditional knowledge base, how to obtain an adequate complete model that can be used for inductive reasoning?

Within a probabilistic framework, conditionals can be quantified and interpreted probabilistically via conditional probabilities:

$$P \models (B|A)[x] \quad \text{iff} \quad P(A) > 0 \text{ and } P(AB) = xP(A)$$

for $x \in [0, 1]$. A *conditional probabilistic knowledge base* is a set $\mathcal{R}^{prob} = \{(B_1|A_1)[x_1], \ldots, (B_n|A_n)[x_n]\}$ of probabilistic conditionals.

Suppose such a conditional probabilistic knowledge base $\mathcal{R}^{prob}$ is given. For instance, $\mathcal{R}^{prob}$ may describe the knowledge available to a physician when he has to make a diagnosis. Or, $\mathcal{R}^{prob}$ may express commonsense knowledge like "*Students are young with a probability of (about) 80 %*" and "*Singles (i.e. unmarried people) are young with a probability of (about) 70 %*", this knowledge being formally expressed by $\mathcal{R}^{prob} = \{(\text{young}\,|\text{student})[0.8], (\text{young}\,|\text{single})[0.7]\}$. Usually, such rule bases represent incomplete knowledge, in that there are a lot of probability distributions apt to represent them. So learning, or inductively representing, respectively, the rules means to take them as a set of conditional constraints and to select a unique probability distribution as a "best" model which can be used for queries and further inferences. Paris [9] investigates several inductive representation techniques and proves that the *principle of maximum entropy, (*ME*-principle)* yields the only method to represent incomplete knowledge in an unbiased way, satisfying a set of postulates describing sound commonsense reasoning. The entropy $H(P)$ of a probability distribution P is defined as

$$H(P) = -\sum_{\omega} P(\omega) \log P(\omega)$$

and measures the amount of indeterminateness inherent in P. Applying the principle of maximum entropy then means to select the unique distribution $P^* = \mathsf{ME}(\mathcal{R}^{prob})$ that maximizes $H(P)$ subject to $P \models \mathcal{R}^{prob}$. In this way, the ME-method ensures that no further information is added, so that the knowledge $\mathcal{R}^{prob}$ is represented most faithfully. $\mathsf{ME}(\mathcal{R}^{prob})$ can be written in the form

$$\mathsf{ME}(\mathcal{R}^{prob})(\omega) = \alpha_0 \prod_{\substack{1 \le i \le n \\ \omega \models A_i B_i}} \alpha_i^{1-x_i} \prod_{\substack{1 \le i \le n \\ \omega \models A_i \overline{B_i}}} \alpha_i^{-x_i} \qquad (1)$$

with the α_i's being chosen appropriately so as to satisfy all of the conditional constraints in $\mathcal{R}^{prob}$ (cf. [10]); $\mathsf{ME}(\mathcal{R}^{prob})$ is called the ME-*representation of* $\mathcal{R}^{prob}$. The ME-principle provides a most convenient and theoretically sound method to represent incomplete probabilistic knowledge[1] and for high-quality probabilistic reasoning (cf. [11]).

A purely probabilistic representation gives precise numerical values to all propositions and conditionals of the underlying language. This can be problematic with respect to two points: First, when the aim is to model subjective beliefs of an expert or an agent, precise probabilities are hard to specify. Subjective probabilities are more or less rough guidelines that are based on an agent's experience. Second, even objective probabilities derived from statistical data may not represent a completely accurate picture of the world. Statistical data can be noisy and only reflect a snapshot of the world, which can be quite accidental. Therefore, in this paper, we are interested in the qualitative knowledge that underlies some given probabilistic information. To represent such qualitative structures, we use *ordinal conditional functions,* OCFs, as introduced by Spohn [7] as a qualitative abstraction of probability functions.

[1] Efficient implementations of ME-systems can be found via `www.informatik.fernuni-hagen.de/pi8/research/projects.html` and `www.pit-systems.de`

Definition 1. *An* ordinal conditional function *(or ranking function)* κ *is a function* $\kappa : \Omega \to \mathbb{N} \cup \{\infty\}$ *with* $\kappa^{-1}(0) \neq \emptyset$.

An OCF κ assigns a *degree of implausibility* (or *ranking value*) to each world ω: The higher $\kappa(\omega)$, the less plausible is ω. A world ω with $\kappa(\omega) = 0$ is regarded as being completely normal (most plausible), and for a consistent modelling, there has to be at least one such world. For formulas $A \in \mathcal{L}$, a ranking is computed via

$$\kappa(A) = \begin{cases} \min\{\kappa(\omega) \mid \omega \models A\} & \text{if } A \text{ is satisfiable} \\ \infty & \text{otherwise} \end{cases}.$$

So we have $\kappa(A \vee B) = \min\{\kappa(A), \kappa(B)\}$ and in particular, $\kappa(A \vee \overline{A}) = 0$. The *belief* in (or *acceptance* of) a formula A is defined as

$$\kappa \models A \quad \text{iff} \quad \kappa(\overline{A}) > 0,$$

i. e., $\kappa(A) = 0$ is necessary but not sufficient to believe A, because $\kappa(\overline{A})$ might be 0 as well; but $\kappa(\overline{A}) > 0$ is sufficient, since it implies $\kappa(A) = 0$.

Similar to the probabilistic framework, conditionals can be quantified. An OCF κ is extended to conditionals by setting

$$\kappa(B|A) = \begin{cases} \kappa(AB) - \kappa(A) & \text{if } \kappa(A) \neq \infty \\ \infty & \text{otherwise} \end{cases},$$

and a conditional is *accepted* by κ,

$$\kappa \models (B|A) \quad \text{iff} \quad \kappa(AB) < \kappa(A\overline{B}) \quad \text{iff} \quad \kappa(\overline{B}|A) > 0.$$

As usual, a proposition A is identified with the conditional $(A|\top)$, hence $\kappa \models (A|\top)$ iff $\kappa(\overline{A}) > \kappa(A) = 0$, in accordance with what was said above.

The acceptance relation for quantified *OCF-conditionals* $(B|A)[m]$ is defined by using the difference between $\kappa(AB)$ and $\kappa(A\overline{B})$:

$$\kappa \models (B|A)[m] \text{ iff } \kappa(AB) + m = \kappa(A\overline{B}) \text{ iff } \kappa(\overline{B}|A) = m,\ m \in \mathbb{N}, m \geq 1. \quad (2)$$

Thus, if $(B|A)$ is believed with a *degree of belief* m then verifying the conditional is m degrees more plausible than falsifying it. So, $\kappa \models (B|A)[1]$ expresses belief in $(B|A)$, but only to the smallest possible degree. For a propositional fact A, this yields

$$\kappa \models A[m] \quad \text{iff} \quad \kappa(\overline{A}) = m.$$

Both qualitative and quantitative OCF-conditionals can be used as default rules for commonsense reasoning [1,11].

Ranking functions provide a perfect framework for qualitative reasoning, as they allow us to handle conditionals in a purely qualitative manner, but also leave room to take more precise, quantitative information into account. However, even numerical information merely expresses an order of magnitude of probabilities; this will be made more precise in the following section.

Moreover, in a qualitative framework with ordinal conditional functions, a similar concept as an ME-representation can be defined in order to express a certain "well-behavedness" of an OCF with respect to a set of OCF-conditionals [11]. We will come back to these issues and present said concept in section 4. In the next section we will first have a look on how to derive an OCF from an empirically obtained probability distribution.

3 Deriving Qualitative Information from Statistical Data

Let P be a probability distribution over $\mathcal{V}$ that could have been collected via a statistical survey. In this paper we are interested in the qualitative structure that underlies the probabilities in P. So we represent P by qualitative probabilities yielding an ordinal conditional function that approximates the quantitative structure in P.

For this reason we start by representing a probability of a specific world ω as polynomial in a fixed base value ε in the spirit of [1]. Using this base representation, the order of magnitude of a probability can be represented only by the corresponding exponents and different probabilities can be compared by these exponents yielding a qualitative abstraction of the original values.

Definition 2. *Let $\varepsilon \in (0,1)$ be a base value to parameterize probabilities. Then a probability value $P(\omega)$ can be expressed as a* polynomial in ε *,*

$$P_\varepsilon(\omega) = a_0\varepsilon^0 + a_1\varepsilon^1 + a_2\varepsilon^2 + \dots \quad ,$$

with appropriate coefficients $a_i \in \mathbb{N}$ respecting $0 \leq a_i < \varepsilon^{-1}$ for all i to match the value $P(\omega)$.

Due to the restriction $0 \leq a_i < \varepsilon^{-1}$ the above definition is sound and uniquely determines a base representation $P_\varepsilon(\omega)$ for given $P(\omega)$ and ε with $P_\varepsilon(\omega) = P(\omega)$.

Example 1. Let $\varepsilon = 0.3$. Then the probability $P(\omega_1) = 0.171$ is written as a polynomial $P_\varepsilon(\omega_1) = 0 \cdot 0.3^0 + 0 \cdot 0.3^1 + 1 \cdot 0.3^{\mathbf{2}} + 3 \cdot 0.3^3$ in ε.

Observe that in the above approach the value of a_0 is always zero, except for the case that the world ω has a probability of 1, which is unlikely the case in real world scenarios. Furthermore the above definition differs from the definition of polynomial base representations in [1] in the sense, that Goldszmidt and Pearl implicitly use negative coefficients for their base representation, representing probabilities as polynomials of the form $P'_\varepsilon(\omega) = 1 - a\varepsilon$ or $P'_\varepsilon(\omega) = a\varepsilon^2 - b\varepsilon^4$. However, an additive representation of positive values like probabilities seems more appropriate for our intentions. Nonetheless, Goldsmizdt and Pearl restrict their attention on qualitative abstractions of probabilities to the case of infinitesimal bases yielding the following definition of a complete translation of all probability values into rankings.

Definition 3 (see [1]). *Let P be a probability distribution and let the probability $P(\omega)$ be written as a polynomial $P'_\varepsilon(\omega)$ in ε with an infinitesimal ε. A ranking function $\kappa_0^P(\omega)$ is defined as follows*

$$\kappa_0^P(\omega) = \begin{cases} \min\{n \in \mathbb{N} \mid \lim_{\varepsilon \to 0} \frac{P'_\varepsilon(\omega)}{\varepsilon^n} \neq 0\} & \text{if } P'_\varepsilon(\omega) > 0 \\ \infty & \text{if } P'_\varepsilon(\omega) = 0 \end{cases}$$

The general idea of the above definition is to capture the most significant term of the base representation of a probability of a world ω, i. e., the first coefficient a_i that differs from zero, and use this value as the rank of ω

$$\kappa_0^P(\omega) = \min\{i \mid a_i \neq 0\}, \quad P_\varepsilon(\omega) = a_0\varepsilon^0 + a_1\varepsilon^1 + \ldots \tag{3}$$

In this paper, we use this idea for a fixed value ε for the base representation and take this value throughout the process of qualitative knowledge discovery as an indicator for the granularity of the qualitative probabilities. Given a fixed base value ε, we determine the most significant term of a base representation with respect to ε and use this value as a rank value for an OCF $\tilde{\kappa}_\varepsilon^P$ as in equation (3). More specifically, let ω be a world and $P(\omega)$ its (empirical) probability. From now on let $\varepsilon \in (0, 1)$ be a fixed base value and let

$$P_\varepsilon(\omega) = a_0\varepsilon^0 + a_1\varepsilon^1 + a_2\varepsilon^2 + \ldots$$

be the base representation of $P(\omega)$ according to Definition 2. We are looking for the first a_i that differs from zero to define the rank of ω:

$$\tilde{\kappa}_\varepsilon^P(\omega) = \min\{i \mid a_i \neq 0\} \quad .$$

Let i satisfy $a_i \neq 0$. Then it holds that

$$P(\omega) \geq a_i\varepsilon^i \geq \varepsilon^i$$

because a_i is a natural number and $a_i > 0$. From this observation, it follows immediately

$$\begin{aligned} & P(\omega) \geq \varepsilon^i \\ \Leftrightarrow\ & \log P(\omega) \geq i \log \varepsilon \\ \Leftrightarrow\ & \frac{\log P(\omega)}{\log \varepsilon} \leq i \quad . \end{aligned}$$

Therefore for the minimal i satisfying $a_i \neq 0$ and so for the rank assigned to ω it follows

$$\tilde{\kappa}_\varepsilon^P(\omega) = \left\lceil \frac{\log P(\omega)}{\log \varepsilon} \right\rceil \tag{4}$$

In general, the function $\tilde{\kappa}_\varepsilon^P$ defined using equation (4) does not satisfy $(\tilde{\kappa}_\varepsilon^P)^{-1}(0) \neq \emptyset$. Therefore, we normalize $\tilde{\kappa}_\varepsilon^P$ by shifting all ranking values appropriately, i. e., by defining $\kappa_\varepsilon^P(\omega) := \tilde{\kappa}_\varepsilon^P(\omega) - c$ with $c = \min\{\tilde{\kappa}_\varepsilon^P(\omega) \mid \omega \in \Omega\}$. Then κ_ε^P defines an ordinal conditional function according to Definition 1. As κ_ε^P is the only ordinal conditional function we are dealing with, we will write just κ for κ_ε^P, when P and ε are clear from context.

Example 2. (Continuing Example 1)
With ε being 0.3, the probability $P(\omega_1) = 0.171$ is written as a polynomial $P_\varepsilon(\omega_1) = 0 \cdot 0.3^0 + 0 \cdot 0.3^1 + 1 \cdot 0.3^{\mathbf{2}} + 3 \cdot 0.3^3$ in ε and therefore $\kappa(\omega_1) = 2$. The probabilities $P(\omega_2) = 0.39$ and $P(\omega_3) = 0.48$ are written as $P_\varepsilon(\omega_2) = 0 \cdot 0.3^0 + 1 \cdot 0.3^{\mathbf{1}} + 1 \cdot 0.3^2$ and $P_\varepsilon(\omega_3) = 0 \cdot 0.3^0 + 1 \cdot 0.3^{\mathbf{1}} + 2 \cdot 0.3^2$, respectively, and so they are both projected to the same ranking value $\kappa(\omega_2) = \kappa(\omega_3) = 1$.

A process of transforming a given probability distribution into a qualitative representation (according to equation (4)) is crucially influenced by the chosen base value ε. It depends on ε how similar some probabilities must be to be projected to the same ranking value. Thus, ε is the parameter that controls the qualitative smoothing of the probabilities. For this reason, an appropriate choice for ε is important for the qualitative modeling since it determines the variation in the resulting ranking values and this way it heavily influences all following calculations based on this values. If the value for ε is close to 1, then even quite similar probabilities will still be projected to different ranking values.

However, a too small value of ε will have the effect that even quite different probabilities will be assigned an identical ranking value. Thus, an unacceptable large amount of information contained in the probabilities will be lost, i. e., the probabilities are smoothed so much that the resulting ranking values do not carry enough information to be useful as a qualitative abstraction.

The following example will illustrate to what degree the choice of ε influences the resulting ranking values.

Example 3. Suppose in our universe are *animals* (A), *fish* (B), *aquatic beings* (C), *objects with gills* (D) and *objects with scales* (E). Table 1 may reflect our observations. Table 2 shows the ranking values that result from different choices of ε.

Table 1. Empirical probabilities for Example 3

ω	*object*	*frequency*	*probability*
ω_1	$abcde$	59	0.5463
ω_2	$abcd\overline{e}$	21	0.1944
ω_3	$a\overline{b}cde$	11	0.1019
ω_4	$a\overline{b}cd\overline{e}$	9	0.0833
ω_5	$abc\overline{d}e$	6	0.0556
ω_6	$abc\overline{d}\overline{e}$	2	0.0185

Choosing $\varepsilon = 0.1$ assigns identical ranking values to ω_1, ω_2 and ω_3 and to ω_4, ω_5 and ω_6, respectively. Mapping the latter ones to the same rank could be acceptable, but mapping the former ones to a common rank is inappropriate, since the probabilities of these worlds cover a (comparative) large range between 0.5463 and 0.1019. Hence, this choice for ε smoothes the probabilities too much, leading to a qualitative abstraction that is so coarse that almost all information of the observed distribution is lost. Choosing $\varepsilon = 0.9$ leads to different ranking value for all ω, although some of the probabilities are quite similar and therefore should not be distinguished in a qualitative setting. Hence, this choice for ε does not seem very appropriate as well because it does not smooth the probabilities effectively. Choosing $\varepsilon = 0.6$ results in a common ranking value for the (comparative) similar probabilities of ω_3 and ω_4. This choice for ε seems to be appropriate to obtain ranking values that form a qualitative representation of the observed probabilities.

Table 2. Ranking values resulting from different choices of ε

	ranking value		
ω	ε =0.1	ε =0.6	ε =0.9
ω_1	1	2	6
ω_2	1	4	16
ω_3	1	5	22
ω_4	2	5	24
ω_5	2	6	28
ω_6	2	8	38

In this very small example, the worlds ω offer quite high probabilities. For this reason, the appropriate value for ε is quite big, too. In a more realistic setting with considerably smaller probabilities, a much smaller value for ε would be chosen.

The parameter ε defines a measure of similarity that is to make probabilities indistinguishable. In principle, it is up to the user to set ε, depending on his point of view, but clustering techniques applied to the logarithmic probabilities may help to find an appropriate ε. A useful heuristic may be to fix a logarithmic similarity α, i.e. probabilities should not be distinguished if their logarithmic distance does not exceed α. Then clusters of logarithmic probabilities with maximal width α are built. A dendrogram computed by, e. g., a complete link clustering procedure (cf. [12]) may provide helpful information for this. Moreover, α should be chosen in such a way that no multiple $k\alpha$ of α falls within one of the clusters. Finally, $\varepsilon = e^{-\alpha}$ may serve to extract ranking infomation from the empirical probabilities. We will illustrate this in our Example 3.

Example 4. Table 3 shows the logarithmic probabilities $\log_e P(\omega)$ of our example. If

Table 3. Logarithmic probabilities

ω	*object*	*frequency*	*probability*	*log. probability*
ω_1	$abcde$	59	0.5463	-0.60
ω_2	$abcd\overline{e}$	21	0.1944	-1.64
ω_3	$a\overline{b}cde$	11	0.1019	-2.28
ω_4	$a\overline{b}cd\overline{e}$	9	0.0833	-2.49
ω_5	$abc\overline{d}e$	6	0.0556	-2.89
ω_6	$abc\overline{d}\overline{e}$	2	0.0185	-3.99

we use a logarithmic similarity $\alpha = 0.5$, then only $P(\omega_3)$ and $P(\omega_4)$ are close enough to be identified, and all multiples of 0.5 discriminate the clusters clearly. Hence $\varepsilon = e^{-0.5} \approx 0.6$ yields an adequate ranking function.

In the next section, we develop an algebraic theory of conditionals, that is used to obtain structural information from such ordinal conditional functions like the one derived above.

4 Conditional Structures and c-Representations

In order to obtain structural information from data, one usually searches for causal relationships by investigating conditional independencies and thus non-interactivity between sets of variables [13,14,15,16]. Some of these algorithms also make use of optimization criteria which are based on entropy [17,18]. Although causality is undoubtedly most important for human understanding, it seems to be too rigid a concept to represent human knowledge in an exhaustive way. For instance, a person suffering from a flu is certainly sick ($P(\text{sick}\,|\text{flu}) = 1$), and they often will complain about headache ($P(\text{headcache}\,|\text{flu}) = 0.9$). Then we have

$$P(\text{headcache}\,|\text{flu}) = P(\text{headcache}\,|\text{flu} \wedge \text{sick}),$$

but we would surely expect

$$P(\text{headcache}\,|\neg\text{flu}) \neq P(\text{headcache}\,|\neg\text{flu} \wedge \text{sick})!$$

Although, from a naïve point of view, the (first) equality suggests a conditional independence between sick and headcache, due to the causal dependency between headcache and flu, the (second) inequality shows this to be (of course) false. Furthermore, a physician might also wish to state some conditional probability involving *sick* and *headache*, so that we would obtain a complex network of rules. Each of these rules will be considered relevant by the expert, but none will be found when searching for conditional independencies! So, what actually are the "structures of knowledge" by which conditional dependencies (not independencies!) manifest themselves in data? What are the "footprints" conditionals leave on probabilities after they have been learned inductively?

A well-known approach to answer this question is *system Z* [1] that builds up a completely specified ranking function from a set of conditionals $\{(B_1|A_1), \ldots, (B_n|A_n)\}$ and yields an inductive reasoning method that satisfies basic properties of default reasoning. In this paper, however, we use *c-representations* for qualitative inductive reasoning that have been developed in [4,11]; all proofs and lots of examples can be found in [11]. This approach follows the same structural lines as ME-reasoning and provides the techniques for model-based inductive reasoning in a qualitative environment the quality of which outperforms *system Z* clearly [19,20].

We first take a structural look on conditionals, bare of numerical values, that is, we focus on sets $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ of unquantified conditionals.

In order to model its non-classical uncertainty, we represent a conditional $(B|A)$ as a three-valued indicator function on worlds

$$(B|A)(\omega) = \begin{cases} 1 & : \quad \omega \models AB \\ 0 & : \quad \omega \models A\overline{B} \\ u & : \quad \omega \models \overline{A} \end{cases}$$

where u stands for *unknown*, following an idea of de Finetti (cf., e. g., [21,22]). Two conditionals are *equivalent* iff they yield the same indicator function, so that $(B|A) \equiv (D|C)$ iff $AB \equiv CD$ and $A\overline{B} \equiv C\overline{D}$.

We generalize this approach by associating to each conditional $(B_i|A_i)$ in $\mathcal{R}$ two abstract symbols $\mathbf{a}_i^+, \mathbf{a}_i^-$, symbolizing a (possibly) positive effect on verifying worlds and a (possibly) negative effect on falsifying worlds:

$$\sigma_i(\omega) = \begin{cases} \mathbf{a}_i^+ & \text{if} \quad \omega \models A_iB_i \\ \mathbf{a}_i^- & \text{if} \quad \omega \models A_i\overline{B_i} \\ 1 & \text{if} \quad \omega \models \overline{A_i} \end{cases} \tag{5}$$

with 1 being the neutral element of the (free abelian) group $\mathfrak{F}_\mathcal{R} = \langle \mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^- \rangle$, generated by all symbols $\mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^-$. The function $\sigma_\mathcal{R} : \Omega \to \mathfrak{F}_\mathcal{R}$, defined by

$$\sigma_\mathcal{R}(\omega) = \prod_{1 \le i \le n} \sigma_i(\omega) = \prod_{\substack{1 \le i \le n \\ \omega \models A_iB_i}} \mathbf{a}_i^+ \prod_{\substack{1 \le i \le n \\ \omega \models A_i\overline{B}_i}} \mathbf{a}_i^- \tag{6}$$

describes the all-over effect of $\mathcal{R}$ on ω. $\sigma_\mathcal{R}(\omega)$ is called the *conditional structure of* ω *with respect to* $\mathcal{R}$.

Example 5. Let $\mathcal{R} = \{(c|a), (c|b)\}$, where A, B, C are bivalued propositional variables with outcomes $\{a, \overline{a}\}, \{b, \overline{b}\}$ and $\{c, \overline{c}\}$, respectively, and let $\mathfrak{F}_\mathcal{R} = \langle \mathbf{a}_1^+, \mathbf{a}_1^-, \mathbf{a}_2^+, \mathbf{a}_2^- \rangle$. We associate $\mathbf{a}_1^+, \mathbf{a}_1^-$ with the first conditional, $(c|a)$, and $\mathbf{a}_2^+, \mathbf{a}_2^-$ with the second one, $(c|b)$. Since $\omega = abc$ verifies both conditionals, we obtain $\sigma_\mathcal{R}(abc) = \mathbf{a}_1^+\mathbf{a}_2^+$. In the same way, e.g., $\sigma_\mathcal{R}(ab\overline{c}) = \mathbf{a}_1^-\mathbf{a}_2^-$, $\sigma_\mathcal{R}(a\overline{b}c) = \mathbf{a}_1^+$ and $\sigma_\mathcal{R}(\overline{a}b\overline{c}) = \mathbf{a}_2^-$.

Let $\hat{\Omega} := \langle \hat{\omega} \mid \omega \in \Omega \rangle$ be the free abelian group generated by all $\omega \in \Omega$, and consisting of all products $\hat{\omega} = {\omega_1}^{r_1} \ldots {\omega_m}^{r_m}$ with $\omega_1, \ldots, \omega_m \in \Omega$ and integers $r_1, \ldots r_m$. Note that, although we speak of *multiplication*, the worlds in such a product are merely juxtaposed, forming a *word* rather than a *product*. With this understanding, a *generalized world* $\hat{\omega} \in \hat{\Omega}$ in which only positive exponents occur simply corresponds to a multi-set of worlds. We will often use fractional representations for the elements of $\hat{\Omega}$, that is, for instance, we will write $\dfrac{\omega_1}{\omega_2}$ instead of $\omega_1\omega_2^{-1}$. Now $\sigma_\mathcal{R}$ may be extended to $\hat{\Omega}$ in a straightforward manner by setting

$$\sigma_\mathcal{R}({\omega_1}^{r_1} \ldots {\omega_m}^{r_m}) = \sigma_\mathcal{R}(\omega_1)^{r_1} \ldots \sigma_\mathcal{R}(\omega_m)^{r_m}$$

yielding a *homomorphism of groups* $\sigma_\mathcal{R} : \hat{\Omega} \to \mathfrak{F}_\mathcal{R}$.

Having the same conditional structure defines an equivalence relation $\equiv_\mathcal{R}$ on $\hat{\Omega}$: $\hat{\omega}_1 \equiv_\mathcal{R} \hat{\omega}_2$ iff $\sigma_\mathcal{R}(\hat{\omega}_1) = \sigma_\mathcal{R}(\hat{\omega}_2)$, i. e. iff $\hat{\omega}_1\hat{\omega}_2^{-1} \in ker\ \sigma_\mathcal{R} := \{\hat{\omega} \in \hat{\Omega} \mid \sigma_\mathcal{R}(\hat{\omega}) = 1\}$. Thus the kernel of $\sigma_\mathcal{R}$ plays an important part in identifying the conditional structure of elements $\hat{\omega} \in \hat{\Omega}$. $ker\ \sigma_\mathcal{R}$ contains exactly all group elements $\hat{\omega} \in \hat{\Omega}$ with a balanced conditional structure, that means, where all effects of conditionals in $\mathcal{R}$ on worlds occurring in $\hat{\omega}$ are completely cancelled. Since $\mathfrak{F}_\mathcal{R}$ is free abelian, no nontrivial relations hold between the different group generators $\mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^-$ of $\mathfrak{F}_\mathcal{R}$, so we have $\sigma_\mathcal{R}(\hat{\omega}) = 1$ iff $\sigma_i(\hat{\omega}) = 1$ for all i, $1 \le i \le n$, and this means

$$ker\ \sigma_\mathcal{R} = \bigcap_{i=1}^{n} ker\ \sigma_i.$$

In this way, each conditional in $\mathcal{R}$ contributes to $ker\ \sigma_{\mathcal{R}}$.

Besides the explicit representation of knowledge by $\mathcal{R}$, also the implicit normalizing constraint $\kappa(\top|\top) = 0$ for ordinal conditional functions has to be taken into account. It is easy to check that $ker\ \sigma_{(\top|\top)} = \hat{\Omega}_0$, with

$$\hat{\Omega}_0 := \{\hat{\omega} = {\omega_1}^{r_1} \cdot \ldots \cdot {\omega_m}^{r_m} \in \hat{\Omega} \mid \sum_{j=1}^{m} r_j = 0\}.$$

Two elements $\hat{\omega}_1 = \omega_1^{r_1} \ldots \omega_m^{r_m}$, $\hat{\omega}_2 = \nu_1^{s_1} \ldots \nu_p^{s_p} \in \hat{\Omega}$ are equivalent modulo $\hat{\Omega}_0$, $\hat{\omega}_1 \equiv_\top \hat{\omega}_2$, iff $\hat{\omega}_1 \hat{\Omega}_0 = \hat{\omega}_2 \hat{\Omega}_0$, i.e. iff $\sum_{1 \leq j \leq m} r_j = \sum_{1 \leq k \leq p} s_k$. This means that $\hat{\omega}_1$ and $\hat{\omega}_2$ are equivalent modulo $\hat{\Omega}_0$ iff they both are a (cancelled) product of the same number of generators, each generator being counted with its corresponding exponent. Set

$$ker_0\ \sigma_{\mathcal{R}} := ker\ \sigma_{\mathcal{R}} \cap \hat{\Omega}_0 = ker\ \sigma_{\mathcal{R} \cup \{(\top|\top)\}}.$$

In the following, if not stated otherwise, we will assume that all ordinal conditional functions are finite, i. e., it is $\kappa(A) \neq \infty$ for every A. For the methods to be described, this is but a technical prerequisite, permitting a more concise presentation of the basic ideas. The general case may be dealt with in a similar manner (cf. [11]). Moreover, in section 5 we will see that we can get rid of all infinite ranking values (which correspond to zero probabilities in the empirical distribution) right from the beginning.

Finite ranking functions κ may be extended easily to homomorphisms $\kappa : \hat{\Omega} \to (\mathbb{Z}, +)$ from $\hat{\Omega}$ into the additive group of integers in a straightforward way by setting

$$\kappa({\omega_1}^{r_1} \ldots {\omega_m}^{r_m}) = r_1 \kappa(\omega_1) + \ldots + r_m \kappa(\omega_m).$$

Definition 4 (Conditional indifference). *Suppose κ is a (finite) ordinal conditional function, and let $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ be a set of conditionals. κ is* (conditionally) indifferent with respect to $\mathcal{R}$ *iff $\kappa(\hat{\omega}_1) = \kappa(\hat{\omega}_2)$, whenever both $\hat{\omega}_1 \equiv_{\mathcal{R}} \hat{\omega}_2$ and $\hat{\omega}_1 \equiv_\top \hat{\omega}_2$ hold for $\hat{\omega}_1, \hat{\omega}_2 \in \hat{\Omega}$.*

If κ is indifferent with respect to $\mathcal{R}$, then it does not distinguish between elements $\hat{\omega}_1 \equiv_\top \hat{\omega}_2$ with the same conditional structure with respect to $\mathcal{R}$. Conversely, any deviation $\kappa(\hat{\omega}) \neq 0$ can be explained by the conditionals in $\mathcal{R}$ acting on $\hat{\omega}$ in a non-balanced way. Note that the notion of indifference only aims at observing conditional structures, without making use of any degrees of belief that are associated with the conditionals.

The following proposition shows, that conditional indifference establishes a connection between the kernels $ker_0\ \sigma_{\mathcal{R}}$ and

$$ker_0\ \kappa := \{\hat{\omega} \in \hat{\Omega}_0 \mid \kappa(\hat{\omega}) = 0\}$$

which will be crucial to elaborate conditional structures:

Proposition 1. *An ordinal conditional function κ is indifferent with respect to a set $\mathcal{R} \subseteq (\mathcal{L}|\mathcal{L})$ of conditionals iff $ker_0\ \sigma_{\mathcal{R}} \subseteq ker_0\ \kappa$.*

If $ker_0\ \sigma_{\mathcal{R}} = ker_0\ \kappa$, then $\kappa(\hat{\omega}_1) = \kappa(\hat{\omega}_2)$ iff $\sigma_{\mathcal{R}}(\hat{\omega}_1) = \sigma_{\mathcal{R}}(\hat{\omega}_2)$, for $\hat{\omega}_1 \equiv_\top \hat{\omega}_2$. In this case, κ completely follows the conditional structures imposed by $\mathcal{R}$ – it observes $\mathcal{R}$ *faithfully*.

The next theorem characterizes indifferent ordinal conditional functions:

Theorem 1. *An ordinal conditional function κ is indifferent with respect to a set $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\} \subseteq (\mathcal{L}|\mathcal{L})$ iff $\kappa(A_i) \neq \infty$ for all i, $1 \leq i \leq n$ and there are rational numbers $\kappa_0, \kappa_1^+, \kappa_1^-, \ldots, \kappa_n^+, \kappa_n^- \in \mathbb{Q}$, such that*

$$\kappa(\omega) = \kappa_0 + \sum_{\substack{1 \leq i \leq n \\ \omega \models A_iB_i}} \kappa_i^+ + \sum_{\substack{1 \leq i \leq n \\ \omega \models A_i\overline{B_i}}} \kappa_i^-, \tag{7}$$

for all $\omega \in \Omega$.

There are striking similarities between (1), (6), and (7). The equations (1) and (7) are both implementations of (6): while in (1) multiplication is used for combining the operands, in (7) it is addition. Furthermore, in (1), the abstract symbols $\mathbf{a}_i^+, \mathbf{a}_i^-$ of (6) have been replaced by the numerical values $\alpha_i^{1-x_i}$ and $\alpha_i^{-x_i}$, respectively (α_0 is simply a normalizing factor). In (7), additive constants κ_i^+, κ_i^- realize the structural effects of conditionals. Both the α_i's and the κ_i's bear crucial conditional information, leaving "footprints" on probabilities resp. ranking values when inductively representing conditionals (also cf. [10]). In [11] it is shown that ordinal conditional functions and probability distributions can be subsumed by the general concept of *conditional valuation functions*.

Example 6. We continue Example 5. Here we observe

$$\sigma_{\mathcal{R}}\left(\frac{abc \cdot \overline{a}\overline{b}\overline{c}}{a\overline{b}c \cdot \overline{a}bc}\right) = \frac{\sigma_{\mathcal{R}}(abc) \cdot \sigma_{\mathcal{R}}(\overline{a}\overline{b}\overline{c})}{\sigma_{\mathcal{R}}(a\overline{b}c) \cdot \sigma_{\mathcal{R}}(\overline{a}bc)} = \frac{\mathbf{a}_1^+\mathbf{a}_2^+ \cdot 1}{\mathbf{a}_1^+ \cdot \mathbf{a}_2^+} = 1,$$

that is, $\dfrac{abc \cdot \overline{a}\overline{b}\overline{c}}{a\overline{b}c \cdot \overline{a}bc} \in ker_0\, \sigma_{\mathcal{R}}$. Then any ordinal conditional function κ that is indifferent with respect $\mathcal{R}$ will fulfill $\kappa\left(\dfrac{abc \cdot \overline{a}\overline{b}\overline{c}}{a\overline{b}c \cdot \overline{a}bc}\right) = 0$, i. e., $\kappa(abc)+\kappa(\overline{a}\overline{b}\overline{c}) = \kappa(a\overline{b}c)+\kappa(\overline{a}bc)$.

In [23], we investigate the exact relationship between *conditional indifference* and *conditional independence* and show that conditional indifference is the strictly more general concept.

Now, in order to obtain a proper representation of a set of conditionals $\mathcal{R}$, we can use the schema (7) and impose the constraints induced by the conditionals in $\mathcal{R}$.

Definition 5 (C-representation 1). *An ordinal conditional function κ is a* c-representation *of a set $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ of conditionals iff κ is indifferent with respect to $\mathcal{R}$ and accepts all conditionals in $\mathcal{R}$, i. e. $\kappa \models \mathcal{R}$.*

For the constraints $\kappa \models (B_i|A_i)$, $1 \leq i \leq n$, to hold, the additive constants κ_i^+, κ_i^- have to satisfy certain relationships which can be checked easily.

Proposition 2. *An ordinal conditional function κ is a* c-representation *of a set $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ of conditionals, iff κ has the form (7) and the κ_i^+, κ_i^-, $1 \leq i \leq n$, fulfill the following inequality:*

$$\kappa_i^- - \kappa_i^+ > \min_{\omega \models A_i B_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j B_j}} \kappa_j^+ + \sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) - \min_{\omega \models A_i \overline{B}_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j B_j}} \kappa_j^+ + \sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) \tag{8}$$

This approach can be generalized in a straightforward manner to handle quantified OCF-conditionals. If $\mathcal{R}^{\mathrm{OCF}} = \{(B_1|A_1)[m_1], \ldots, (B_n|A_n)[m_n]\}$ is a set of quantified OCF-conditionals, then we denote by $\mathcal{R} = \{(B_1|A_1), \ldots, (B_n|A_n)\}$ its corresponding set of purely qualitative conditinals.

Definition 6 (C-representation 2). *An ordinal conditional function κ is a* c-representation *of a set $\mathcal{R}^{\mathrm{OCF}} = \{(B_1|A_1)[m_1], \ldots, (B_n|A_n)[m_n]\}$ of quantified OCF-conditionals iff κ is indifferent with respect to $\mathcal{R}$ and accepts all conditionals in $\mathcal{R}^{\mathrm{OCF}}$, i. e. $\kappa \models \mathcal{R}^{\mathrm{OCF}}$.*

According to (2), the constraints imposed by $\kappa \models (B_i|A_i)[m_i]$ can be handled in a way similar to the purely qualitative case.

Proposition 3. *An ordinal conditional function κ is a* c-representation *of a set $\mathcal{R}^{\mathrm{OCF}} = \{(B_1|A_1)[m_1], \ldots, (B_n|A_n)[m_n]\}$ of quantified OCF-conditionals, iff κ has the form (7) and the κ_i^+, κ_i^-, $1 \leq i \leq n$, fulfill the following inequality:*

$$\kappa_i^- - \kappa_i^+ = m_i + \min_{\omega \models A_i B_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j B_j}} \kappa_j^+ + \sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) - \min_{\omega \models A_i \overline{B}_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j B_j}} \kappa_j^+ + \sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) \tag{9}$$

For the sake of informational economy, the difference $\kappa_i^- - \kappa_i^+$ reflecting the amount of distortion imposed by a conditional belief should be minimal. A reasonable approach to obtain "small" c-representations κ is to set $\kappa_i^+ = 0$ and to choose κ_i^- minimal, in accordance with (8) resp. (9). This simplifies the reasoning with c-representations a lot. The schema (7) shrinks to

$$\kappa(\omega) = \sum_{\substack{1 \leq i \leq n \\ \omega \models A_i \overline{B_i}}} \kappa_i^-, \ \omega \in \Omega, \tag{10}$$

as for consistent sets of conditionals the normalizing constant κ_0 is always zero, and the inequalities (8) now read

$$\kappa_i^- > \min_{\omega \models A_i B_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) - \min_{\omega \models A_i \overline{B}_i} \Big(\sum_{\substack{j \neq i \\ \omega \models A_j \overline{B}_j}} \kappa_j^- \Big) \tag{11}$$

However, different from the ME-principle in the probabilistic case, even minimal c-representations are not uniquely determined. It is still an open problem of research to specify conditions for unique c-representations. For the knowledge discovery problem dealt with in this paper, this is not a severe problem, as the ranking function is not searched for, but is derived from the given empirical distribution.

In summary, any ordinal conditional function κ that is indifferent with respect to a set of conditionals $\mathcal{R}^{\mathrm{OCF}}$ follows the conditional structures that are imposed by the conditionals in $\mathcal{R}$ onto the worlds and is thus most adequate to represent ordinal conditional knowledge.

In the following, we will put these ideas in formal, algebraic terms and prepare the theoretical grounds for the data mining techniques to be presented in this paper.

5 Discovering Structural Information

In this section, we will describe our approach to knowledge discovery which is based on the algebraic theory of conditionals sketched above. More precisely, we will show how to compute sets $\mathcal{R}$, or $\mathcal{R}^{\mathrm{OCF}}$, respectively, of (quantified) default rules that are apt to generate some given (finite) ordinal conditional function κ^P that is indifferent with respect to $\mathcal{R}$, respectively $\mathcal{R}^{\mathrm{OCF}}$. κ^P has been chosen to represent the observed statistical data P, as has been described in Section 3. More details and all proofs can be found in [11]; a generalization to multivalued variables (instead of bivalued variables) is straightforward.

In our scenario, an empirically obtained probability distribution P is given that may simply consist of relative frequencies. Usually, the aim of a data mining task is to compute a set of probabilistic rules $\mathcal{R}^{prob} = \{(B_1|A_1)[x_1], \ldots, (B_n|A_n)[x_n]\}$, such that this set predicts P best. This task was handled in [5] and also uses the algebraic theory of conditionals sketched above. The problem with the approach of [5] is that usually the empirically obtained probability distribution P is noisy and one can not find an appropriate (and particularly compact) set of probabilistic rules $\mathcal{R}^{prob}$ that explains the observed P. The set of computed probabilistic rules tends to be large and the rules are getting too specific to be helpful in a general context. On this account we present an alternative approach to knowledge discovery that also makes use of the algebraic theory of conditionals of [11] but is based on a representation of P by qualitative probabilities, i. e. by rankings.

As a first step, the probability distribution P is qualified in a sense, that we compute its ranking representation κ^P regarding equation (4). By doing this, we fuse several similar probabilities, that should not be distinguished in a qualitative setting, to one rank value of the obtained ranking function. Therefore we minimize the noise, that could be present in the original distribution P, obtaining a qualitative representation. We can now use the formalism of the algebraic theory of conditionals to compute a set $\mathcal{R}^{\mathrm{OCF}} = \{(B_1|A_1)[m_1], \ldots, (B_n|A_n)[m_n]\}$ of OCF-conditionals, that best explains κ^P, i. e., that is (in the best case) a faithful representation of κ^P. More precisely, we are looking for a set $\mathcal{R}$ of (unquantified) conditionals, such that κ^P is indifferent with

respect to $\mathcal{R}$, i. e., $ker_0\ \sigma_\mathcal{R} \subseteq ker_0\ \kappa^P$ by Proposition 1. Ideally, we would have κ^P to represent $\mathcal{R}$ faithfully, that is,

$$\kappa^P \models \mathcal{R} \text{ and } ker_0\ \kappa^P = ker_0\ \sigma_\mathcal{R} \tag{12}$$

This means κ^P is indifferent with respect to $\mathcal{R}$, and no equation $\kappa^P(\hat{\omega}) = 0$ is fulfilled accidentally, but any of these equations is induced by $\mathcal{R}$.

Finally, we can assign rankings to these conditionals, derived immediately from κ^P thus obtaining a set $\mathcal{R}^{\mathrm{OCF}}$ of OCF-conditionals.

Under the assumption of faithfulness, the structures of the conditionals in $\mathcal{R}$ become manifest in the elements of $ker_0\ \kappa^P$, that is, in elements $\hat{\omega} \in \hat{\Omega}$ with $\kappa^P(\hat{\omega}) = 0$. As a further prerequisite, we will assume that this knowledge inherent to κ^P is representable by a set of single-elementary conditionals. This restriction is not too hard, because single-elementary conditionals are expressive enough to represent most commonsense knowledge. As our approach will work for any given ranking function κ, we omit the superscript P in this section.

So assume $\mathcal{R}^{\mathrm{OCF}} = \{(b_1|A_1)[m_1], \ldots, (b_n|A_n)[m_n]\}$ is an existing, but hidden set of single-elementary conditionals, such that (12) holds. Let us further suppose that $ker_0\ \kappa$ (or parts of it) is known from exploiting numerical relationships. Since conditional indifference is a structural notion, the quantifications m_i of the conditionals will not be needed in what follows. Let $\sigma_\mathcal{R} : \hat{\Omega} \to \mathfrak{F}_\mathcal{R} = \langle \mathbf{a}_1^+, \mathbf{a}_1^-, \ldots, \mathbf{a}_n^+, \mathbf{a}_n^- \rangle$ denote a conditional structure homomorphism with respect to $\mathcal{R}$.

Besides conditional structures, a further notion which is crucial to study and exploit conditional interactions is that of subconditionals: $(D|C)$ is called a *subconditional* of $(B|A)$, and $(B|A)$ is a *superconditional* of $(D|C)$, written as $(D|C) \sqsubseteq (B|A)$, iff $CD \models AB$ and $C\overline{D} \models A\overline{B}$, that is, iff all worlds verifying (falsifying) $(D|C)$ also verify (falsify) $(B|A)$. For any two conditionals $(B|A), (D|C) \in (\mathcal{L}|\mathcal{L})$ with $ABC\overline{D} \equiv A\overline{B}CD \equiv \bot$, the supremum $(B|A) \sqcup (D|C)$ in $(\mathcal{L}|\mathcal{L})$ with respect to $\sqsubseteq$ exists and is given by

$$(B|A) \sqcup (D|C) \equiv (AB \vee CD|A \vee C)$$

(cf. [11]). In particular, for two conditionals $(B|A), (B|C)$ with the same consequent, we have

$$(B|A) \sqcup (B|C) \equiv (B|A \vee C)$$

The following lemma provides an easy characterization for the relation $\sqsubseteq$ to hold between single-elementary conditionals:

Lemma 1. *Let $(b|A)$ and $(d|C)$ be two single-elementary conditionals. Then $(d|C) \sqsubseteq (b|A)$ iff $C \models A$ and $b = d$.*

This lemma may be generalized slightly to hold for conditionals $(b|A)$ and $(d|C)$ where A and C are disjunctions of conjunctions of literals not containing b and d, respectively.

From (5), Definition 4 and Proposition 1, it is clear that in an inductive reasoning process such as a propagation that results in an indifferent representation of conditional knowledge $\mathcal{R}$, all subconditionals of conditionals in $\mathcal{R}$ also exert the same effects on possible worlds as the corresponding superconditionals. The basic idea is to start with

most basic conditionals, and to generalize them step-by-step to superconditionals in accordance with the conditional structure revealed by $ker_0\ \kappa$. From a theoretical point of view, the most adequate candidates for rules to start with are *basic single-elementary conditionals*, which are single-elementary conditionals with antecedents of maximal length:

$$\psi_{v,l} = (v \mid C_{v,l}) \tag{13}$$

where v is a value of some variable $V \in \mathcal{V}$ and $C_{v,l}$ is an elementary conjunction consisting of literals involving all variables from $\mathcal{V}$ except V. It is clear that considering all such conditionals is intractable, but we are still on theoretical grounds, so let us assume for the moment we could start with the set

$$\mathcal{B} = \{\psi_{v,l} \mid v \in \mathcal{V}, l \text{ suitable}\}$$

of all basic single-elementary conditionals in $(\mathcal{L}|\mathcal{L})$, and let $\mathfrak{F}_\mathcal{B} = \langle \mathbf{b}^+_{v,l}, \mathbf{b}^-_{v,l} \rangle_{v,l}$ be the free abelian group corresponding to $\mathcal{B}$ with conditional structure homomorphism $\sigma_\mathcal{B} : \hat{\Omega} \to \mathfrak{F}_\mathcal{B}$. Note that $\sigma_\mathcal{B}$ and $\mathfrak{F}_\mathcal{B}$ are known, whereas $\sigma_\mathcal{R}$ and $\mathfrak{F}_\mathcal{R}$ are not. We only know the kernel, $ker_0\ \sigma_\mathcal{R}$, of $\sigma_\mathcal{R}$, which is, by assuming faithfulness (12), the same as the kernel, $ker_0\ \kappa$, of κ. Now, to establish a connection between what is obvious ($\mathcal{B}$) and what is searched for ($\mathcal{R}$), we define a homomorphism $g : \mathfrak{F}_\mathcal{B} \to \mathfrak{F}_\mathcal{R}$ via

$$g(\mathbf{b}^\pm_{v,l}) := \prod_{\substack{1 \le i \le n \\ \psi_{v,l} \sqsubseteq (b_i \mid A_i)}} \mathbf{a}^\pm_i = \prod_{\substack{1 \le i \le n \\ b_i = v, C_{v,l} \models A_i}} \mathbf{a}^\pm_i, \tag{14}$$

where the second equality holds due to Lemma 1. g uses the subconditional-relationship in collecting for each basic conditional in $\mathcal{B}$ the effects of the corresponding superconditionals in $\mathcal{R}$. Actually, g is a "phantom" which is not explicitly given, but only assumed to exist. Its crucial meaning for the knowledge discovery task is revealed by the following theorem:

Theorem 2. *Let $g : \mathfrak{F}_\mathcal{B} \to \mathfrak{F}_\mathcal{R}$ be as in (14). Then*

$$\sigma_\mathcal{R} = g \circ \sigma_\mathcal{B}$$

In particular, $\hat{\omega} \in ker_0\ \sigma_\mathcal{R} = ker_0\ \kappa$ iff $\hat{\omega} \in \hat{\Omega}_0$ and $\sigma_\mathcal{B}(\hat{\omega}) \in ker\ g$.

This means, that numerical relationships observed in κ (and represented by elements of $ker_0\ \kappa$) translate into group theoretical equations modulo the kernel of g.

Proposition 4. *Let $\hat{\omega} = \omega_1^{r_1} \cdot \ldots \cdot \omega_m^{r_m} \in \hat{\Omega}_0$. Then $\sigma_\mathcal{B}(\omega_1^{r_1} \cdot \ldots \cdot \omega_m^{r_m}) \in ker\ g$ iff for all literals v in $\mathcal{L}$,*

$$\prod_{C_{v,l}} \prod_{\substack{1 \le k \le m \\ \omega_k \models C_{v,l}\, v}} (\mathbf{b}^+_{v,l})^{r_k},\ \prod_{C_{v,l}} \prod_{\substack{1 \le k \le m \\ \omega_k \models C_{v,l}\, \overline{v}}} (\mathbf{b}^-_{v,l})^{r_k} \in ker\ g. \tag{15}$$

So each (generating) element of $ker_0\ \sigma_\mathcal{R}$ gives rise to an equation modulo $ker\ g$ for the generators $\mathbf{b}^+_{v,l}, \mathbf{b}^-_{v,l}$ of $\mathfrak{F}_\mathcal{B}$. Moreover, Proposition 4 allows us to split up equations modulo $ker_0\ g$ to handle each literal separately as a consequent of conditionals, and

to separate positive from negative effects. These separations are possible due to the property of the involved groups of being free abelian, and they are crucial to disentangle conditional interactions (cf. also [11]).

Now the aim of our data mining procedure can be made more precise: We are going to define a finite sequence of sets $\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \ldots$ of conditionals approximating $\mathcal{R}$, in the sense that

$$ker_0\ \sigma_{\mathcal{S}^{(0)}} \subseteq ker_0\ \sigma_{\mathcal{S}^{(1)}} \subseteq \ldots \subseteq ker_0\ \sigma_{\mathcal{R}} = ker_0\ \kappa \tag{16}$$

The set $\mathcal{B}$ of basic single elementary conditionals proves to be an ideal starting point $\mathcal{S}^{(0)}$:

Lemma 2. *$\sigma_{\mathcal{B}}$ is injective, i.e. $ker_0\ \sigma_{\mathcal{B}} = \{1\}$.*

So $\sigma_{\mathcal{B}}$ provides the most finely grained conditional structure on $\hat{\Omega}$: No different elements $\hat{\omega}_1 \neq \hat{\omega}_2$ are equivalent with respect to $\mathcal{B}$.

Step by step, the relations mod $ker\ g$ holding between the group elements are exploited with the aim to construct $\mathcal{S}^{(t+1)}$ from $\mathcal{S}^{(t)}$ by eliminating or joining conditionals by $\sqcup$, in accordance with the equations modulo $ker\ g$ (i. e., by assumption, with the numerical relationships found in κ). Each $\mathcal{S}^{(t)}$ is assumed to be a set of conditionals $\phi^{(t)}_{v,j}$ with a single literal v in the conclusion, and the antecedent $D^{(t)}_{v,j}$ of $\phi^{(t)}_{v,j}$ is a disjunction of elementary conjunctions not mentioning the variable V. Let $\mathfrak{F}_{\mathcal{S}^{(t)}} = \langle \mathbf{s}^{(t)\,+}_{v,j}, \mathbf{s}^{(t)\,-}_{v,j} \rangle_{v,j}$ be the free abelian group associated with $\mathcal{S}^{(t)}$, and $\sigma_{\mathcal{S}^{(t)}} : \hat{\Omega} \to \mathfrak{F}_{\mathcal{S}^{(t)}}$ the corresponding structure homomorphism; let $g^{(t)} : \mathfrak{F}_{\mathcal{S}^{(t)}} \to \mathfrak{F}_{\mathcal{R}}$ be the homomorphism defined by

$$g^{(t)}(\mathbf{s}^{(t)\,\pm}_{v,j}) = \prod_{\substack{1 \leq i \leq n \\ v = b_i, D^{(t)}_{v,j} \models A_i}} \mathbf{a}_i^{\pm}$$

such that $g^{(t)} \circ \sigma_{\mathcal{S}^{(t)}} = \sigma_{\mathcal{R}}$. Let $\equiv_{g^{(t)}}$ denote the equivalence relation modulo $ker\ g^{(t)}$, i. e., $\mathbf{s}_1 \equiv_{g^{(t)}} \mathbf{s}_2$ iff $g^{(t)}(\mathbf{s}_1) = g^{(t)}(\mathbf{s}_2)$ for any two group elements $\mathbf{s}_1, \mathbf{s}_2 \in \mathfrak{F}_{\mathcal{S}^{(t)}}$. In the following, for ease of notation, we will omit the $+, -$ superscripts on group generators; this is justified, since, by Proposition 4, only one $\{+, -\}$-type of generators is assumed to occur in the equations to be dealt with in the sequel. It is clear that all equations can be transformed such that on either side, only generators with positive exponents occur.

The basic type of equation that arises from $ker_0\ \kappa$ by applying Theorem 2 and the faithfulness assumption (12) is of the form

$$\mathbf{s}^{(t)}_{v,j_0} \equiv_{g^{(t)}} \mathbf{s}^{(t)}_{v,j_1} \ldots \mathbf{s}^{(t)}_{v,j_m} \tag{17}$$

To obtain the new set $\mathcal{S}^{(t+1)}$ by solving this equation, the following steps have to be done:

1. eliminate $\phi^{(t)}_{v,j_0}$ from $\mathcal{S}^{(t)}$;
2. replace each $\phi^{(t)}_{v,j_k}$ by $\phi^{(t+1)}_{v,j_k} = \phi^{(t)}_{v,j_0} \sqcup \phi^{(t)}_{v,j_k}$ for $1 \leq k \leq m$.

3. retain all other $\phi_{w,l}^{(t)}$ in $\mathcal{S}^{(t)}$.

This also includes the case $m = 0$, i.e. $\phi_{v,j_0}^{(t)} \equiv_{g^{(t)}} 1$; in this case, Step 2 is vacuous and therefore is left out.

It can be shown (cf. [11]) that

$$g^{(t+1)} \circ \sigma_{\mathcal{S}^{(t+1)}} = \sigma_{\mathcal{R}}$$

and hence

$$ker_0\ \sigma_{\mathcal{S}^{(t)}} \subseteq ker_0\ \sigma_{\mathcal{S}^{(t+1)}} \subseteq ker_0\ \sigma_{\mathcal{R}}$$

as desired. Moreover, $ker\ g^{(t+1)}$ can be obtained directly from $ker\ g^{(t)}$ by straightforward modifications. Since the considered equation has been solved, it can be eliminated, and other equations may simplify.

Now, that the theoretical background and the basic techniques have been described, we will turn to develop an algorithm for conditional knowledge discovery.

6 Learning Default Rules from Data

In this section, we will describe an adjusted version of the *CKD*-algorithm (= *Conditional Knowledge Discovery*) for the determination of default rules from qualitative approximations of statistical data. This algorithm is sketched in Figure 2. The original *CKD*-algorithm for mining probabilistic conditionals from statistical data has been implemented in the CONDOR-system (for an overview, cf. [24]). The resulting set of default rules or OCF-conditionals will reveal relevant relationships and may serve to represent inductively the corresponding ordinal conditional function faithfully.

A problem that has already been mentioned but postponed in section 5 is that the set $\mathcal{B}$ of *all* basic single elementary conditionals is virtually unmanageable. Therefore it cannot be used as an adequate starting set in the algorithm. Another problem emerges from the frequency distributions calculated from a data set. In a realistic setting, these distributions are sparse, i. e., they deliver zero values for many worlds. Hence, the probability value of these worlds is zero as well and according to Definition 3, a world with a zero probability is assigned an infinite ranking value. Besides calculational difficulties, the correct interpretation of such worlds, which have not been observed in the analyzed data and therefore have a frequency of zero, is not clear: On the one hand, these worlds might just have *not been captured* when recorded the data; perhaps because the amount of recorded data was not large enough and they have merely been missed. In this case, assigning these worlds a zero probability would be misleading. On the other hand, these worlds might *not exist* at all (and could therefore not have been recorded), so a zero probability would be completely justified; but this could never be assured by pure observation. The problem of zero probabilities is addressed more deeply in [5].

Both of these problems – the exponential complexity of the ideal conditional starter set and the sparse and mostly incomplete knowledge provided by statistical data – can be solved in our framework in the following way: The zero values in an observed frequency distribution are taken to be *unknown, but equal* probabilities, that is, they are treated as non-knowledge without structure. More exactly, let P be the frequency distribution computed from the set of data under consideration. Then, for each two worlds

Algorithm CKD for OCFs
(Conditional Knowledge Discovery)

Input A probability distribution P obtained from statistical data, (only explicitly listing those entries with positive probabilities) together with information on variables and appertaining values and an abstraction parameter $\varepsilon \in (0, 1)$

Output A set of OCF-conditionals (default rules)

Begin
% Qualitative representation
Calculate the ranking value $\tilde{\kappa}_\varepsilon^P(\omega)$ for each input value $P(\omega)$;
Normalize $\tilde{\kappa}_\varepsilon^P$ to obtain the ordinal conditional function κ_ε^P;

% CKD Initialization
Compute the *basic tree of conjunctions*;
Compute the list *NC* of *null-conjunctions*;
Compute the set $\mathcal{S}^{(0)}$ of *basic rules*;
Compute $ker_0\ \kappa_\varepsilon^P$;
Compute $ker\ g$;
Set $\mathcal{K} := ker\ g$;
Set $\mathcal{S} := \mathcal{S}^{(0)}$;

% CKD Main loop
While equations of type (17) are in $\mathcal{K}$ **Do**
Choose $gp \in \mathcal{K}$ of type (17);
Modify (and compactify) $\mathcal{S}$;
Modify (and reduce) $\mathcal{K}$;

% Present results
Calculate the degrees of belief of the conditionals in $\mathcal{S}$;
Return $\mathcal{S}$ and appertaining degrees of belief;
End.

Fig. 2. The CKD-algorithm for OCFs

ω_1, ω_2 not occurring in the database and thus being assigned an unknown but equal probability, we have $P(\omega_1) = P(\omega_2)$; with κ^P being the corresponding ordinal conditional function, this leads to $\kappa^P(\omega_1) = \kappa^P(\omega_2)$ and hence $\frac{\omega_1}{\omega_2} \in ker_0\ \kappa^P$. In this way, all these so-called *null-worlds* contribute to $ker_0\ \kappa^P$, and their structure may be theoretically exploited to shrink the starting set of conditionals in advance.

In order to represent missing information in a most concise way, *null-conjunctions* (i. e. elementary conjunctions with frequency 0) have to be calculated as disjunctions of null-worlds. To this end, the *basic tree of conjunctions* is built up. Its nodes are labelled by the names of variables, and the outgoing edges are labelled by the corresponding

values, or literals, respectively. The labels of paths going from the root to nodes define elementary conjunctions. So, the leaves of the tree either correspond to complete conjunctions occurring in the database, or to null-conjunctions. These null-conjunctions are collected and aggregated to define a set *NC* of most concise conjunctions of ranking value ∞.

Now we are able to set up a set $\mathcal{S}^{(0)}$ of *basic rules* also with the aid of tree-like structures. First, it is important to observe that, due to Proposition 4, conditionals may be separately dealt with according to the literal occurring in their consequents. So $\mathcal{S}^{(0)}$ consists of sets $\mathcal{S}^{(0,v)}$ of conditionals with consequent v, for each value v of each variable $V \in \mathcal{V}$. Basically, the full trees contain all basic single-elementary conditionals from $\mathcal{B}$, but the trees are pruned with the help of the set *NC* of null-conjunctions. The method to shorten the premises of the rules is the same as has been developed in the previous section with finite ranking values, except that now appropriate modifications have to be anticipated, in order to be able to work with a set of rules of acceptable size right from the beginning.

Now, that the missing values in the frequency distribution corresponding to infinite degrees of disbelief have been absorbed by the shortened basic rules, we explore the finite rankings derived from P to set up $ker_0\ \kappa^P$. Usually, numerical relationships $\kappa^P(\hat{\omega}) = 0$ induced by single-elementary rules can be found between neighboring complete conjunctions (i.e. complete conjunctions that differ in exactly one literal). We construct a *neighbor graph* from κ^P, the vertices of which are the non-∞-worlds, labelled by their finite ranking values, and with edges connecting any two neighbors. Then any such relationship $\kappa^P(\hat{\omega}) = 0$ corresponds to a cycle of even length (i. e. involving an even number of vertices) in the neighbor graph, such that the alternating sum built from the values associated with the vertices, with alternating coefficients $+1$ and -1 according to the order of vertices in the cycle, amounts to 0. Therefore, the search for numerical relationships holding in κ^P amounts to searching for cycles with sum 0 in the neighbor graph.

At this point, an important advantage of using qualitative probabilities, i. e., ranking values, becomes clear: Because the ranking values are discrete values, we can demand that the vertices of a cycle must sum up to *exactly* zero. In the approach of [5] that uses the empirically obtained probabilities directly, one can only demand that vertices of a cycle must *approximately* fulfill the corresponding equation, because equality can usually not be reached when calculating with the exact probabilities, i. e., with continuous values. So in the approach of [5] the important step of exploring the numerical relationships depends implicitly on the notion of "approximately". But by using an (appropriate) explicit parameter ε for the qualitative abstraction of the original probabilities, the search for numerical relationships is defined precisely.

Finally, as the last step of the initialization, $ker\ g$ has to be computed from $ker_0\ \kappa^P$ with respect to the set $\mathcal{S}^{(0)}$ of conditionals, as described in the previous section.

In the main loop of the algorithm *CKD*, the sets $\mathcal{K}$ of group elements and $\mathcal{S}$ of conditionals are subject to change. In the beginning, $\mathcal{K} = ker\ g$ and $\mathcal{S} = \mathcal{S}^{(0)}$; in the end, $\mathcal{S}$ will contain the discovered conditional relationships. More detailed, the products in $\mathcal{K}$ which correspond to equations of type (17) are used to simplify the set $\mathcal{S}$. The modified conditionals induce in turn a modification of $\mathcal{K}$, and this is repeated as long as elements

yielding equations of type (17) can be found in $\mathcal{K}$. Note that no ranking values are used in this main loop – only structural information (derived from numerical information) is processed. It is only afterwards, that the ranking values of the conditionals in the final set $\mathcal{S}$ are computed from κ^P, and the OCF-conditionals (default rules) are returned.

Although equations of type (17) are the most typical ones, more complicated equations may arise, which need further treatment. The techniques described above, however, are basic to solving *any* group equation. More details will be published in a forthcoming paper. But in many cases, we will find that all or nearly all equations in $ker\ g$ can be solved successfully and hence can be eliminated from $\mathcal{K}$.

We will illustrate our method by the following example.

Example 7. (Continuing Example 3)
From the observed probabilities, we calculate qualitative probabilities, using $\varepsilon = 0.6$ as base value. We adjust the calculated qualitative probabilities by subtracting the normalization constant $c = 2$, so that the lowest ranking becomes 0. This gives us the ranking values $\kappa^P(\omega)$ that define the ordinal conditional function κ^P, as can be seen from Table 4.

Table 4. Empirical probabilities and corresponding ranking values

object	*frequency*	*probability*	*rank*
$abcde$	59	0.5463	0
$abcd\overline{e}$	21	0.1944	2
$a\overline{b}cde$	11	0.1019	3
$a\overline{b}cd\overline{e}$	9	0.0833	3
$abc\overline{d}e$	6	0.0556	4
$abc\overline{d}\overline{e}$	2	0.0185	6

The set of *null-conjunctions* is calculated as $NC = \{\overline{a}, \overline{c}, \overline{b}\,\overline{d}\}$ – no object matching any one of these partial descriptions occurs in the data base. These null-conjunctions are crucial to set up a starting set $\mathcal{B}$ of basic rules of feasible size:

$$\begin{aligned}\mathcal{B} = \{\quad & \phi_{b,1} = (b|acde) && \phi_{d,1} = (d|abce)\\ & \phi_{b,2} = (b|acd\overline{e}) && \phi_{d,2} = (d|abc\overline{e})\\ & \phi_{b,3} = (b|\overline{d}) && \phi_{d,3} = (d|\overline{b})\\ & \phi_{e,1} = (e|abcd) && \phi_{a,1} = (a|\top)\\ & \phi_{e,2} = (e|abc\overline{d}) &&\\ & \phi_{e,3} = (e|a\overline{b}cd) && \phi_{c,1} = (c|\top)\ \}\end{aligned}$$

So, the missing information reflected by the set NC of null-conjunctions helped to shrink the starting set $\mathcal{B}$ of rules from $5 \cdot 2^4 = 80$ basic single-elementary rules to only 11 conditionals. The next step is to analyze numerical relationships. In this example, we find two numerical relationships between neighboring worlds that are balanced:

$$\kappa^P(a\overline{b}cde) = \kappa^P(a\overline{b}cd\overline{e}) \quad \text{and} \quad \kappa^P(abcde) - \kappa^P(abcd\overline{e}) = \kappa^P(abc\overline{d}e) - \kappa^P(abc\overline{d}\overline{e})$$

At this point, it becomes clear how crucial an appropriate choice for ε is. If ε had been chosen too high, e. g. $\varepsilon = 0.9$ as in Example 3, then the neighboring worlds ω_3 and ω_4 would have been assigned different ranking values, so the first numerical relationship would not hold. Thus an important piece of structural information would have been missed. On the other hand, if ε had been chosen much too small, e. g. $\varepsilon = 0.01$, then all worlds would have been projected to the same ranking value. Thus relationships between all neighboring worlds would have been established, leading to no useful results.

Continuing the example, the first relationship can be translated into the following structural equations by using $\sigma_{\mathcal{B}}$, according to Theorem 2:

$$\mathbf{b}^{+}_{a,1}\mathbf{b}^{-}_{b,1}\mathbf{b}^{+}_{c,1}\mathbf{b}^{+}_{d,3}\mathbf{b}^{+}_{e,3} \equiv_g \mathbf{b}^{+}_{a,1}\mathbf{b}^{-}_{b,2}\mathbf{b}^{+}_{c,1}\mathbf{b}^{+}_{d,3}\mathbf{b}^{-}_{e,3}$$
$$\Rightarrow \mathbf{b}^{-}_{b,1} \equiv_g \mathbf{b}^{-}_{b,2} \text{ and } \mathbf{b}^{+}_{e,3} \equiv_g \mathbf{b}^{-}_{e,3} \equiv_g 1$$

So $\phi_{b,1}$ and $\phi_{b,2}$ are joined to yield $(b|acd)$, and $\phi_{e,3}$ is eliminated. In a similar way, by exploiting the second relationship in κ^P, we obtain $\mathbf{b}^{\pm}_{d,1} \equiv \mathbf{b}^{\pm}_{d,2}$ and $\mathbf{b}^{\pm}_{e,1} \equiv \mathbf{b}^{\pm}_{e,2}$, that is, the corresponding conditionals have to be joined. As a final output, the CKD algorithm returns the set of conditionals that is shown in Table 5.

Table 5. Conditionals calculated by the CKD algorithm

conditional	*empirical probability*	*degree of belief*
$(a\|\top)$	1	∞
$(b\|\overline{d})$	1	∞
$(b\|acd)$	0.80	3
$(e\|abc)$	0.74	2
$(c\|\top)$	1	∞
$(d\|\overline{b})$	1	∞
$(d\|abc)$	0.91	4

All these conditionals are accepted in κ^P. For each of them the degree of belief regarding κ^P can be stated as well as the probability regarding the observed distribution P. So all objects in our universe are aquatic animals which are fish or have gills. Aquatic animals with gills are mostly fish (with a degree of belief 3 and a probability of 0.80), aquatic fish usually have gills (with a degree of belief 4 and a probability of 0.91) and scales (with a degree of belief 2 and a probability of 0.74).

Furthermore, an approximated probability based on the ranking values can be calculated for each conditional $(B|A)$. Because the ranking values are determined according to equation (4), each probability $P(\omega)$ is qualitatively approximated by its corresponding ranking value, so we have:

$$P(\omega) \approx \varepsilon^{\kappa^P(\omega)} \tag{18}$$

By taking into consideration the equation

$$P(B|A) = \frac{1}{\frac{P(A)}{P(AB)}} = \frac{1}{\frac{P(AB)+P(A\overline{B})}{P(AB)}} = \frac{1}{1+\frac{P(A\overline{B})}{P(AB)}} \quad ,$$

Table 6. Conditionals and their approximated probabilities

conditional	*degree of belief*	*approx. probability*
$(a\|\top)$	∞	1
$(b\|\overline{d})$	∞	1
$(b\|acd)$	3	0.82
$(e\|abc)$	2	0.74
$(c\|\top)$	∞	1
$(d\|\overline{b})$	∞	1
$(d\|abc)$	4	0.89

we can approximate the probability of a conditional by its degree of belief[2] m:

$$P(B|A) \approx \frac{1}{1+\frac{\varepsilon^{\kappa^P(A\overline{B})}}{\varepsilon^{\kappa^P(AB)}}} = \frac{1}{1+\varepsilon^{\kappa^P(A\overline{B})-\kappa^P(AB)}} = \frac{1}{1+\varepsilon^m} \tag{19}$$

Example 8. (Continuing Example 7)
The application of formula (19) results in approximated conditional probabilities, listed in Table 6. Compared to the exactly calculated empirical probabilities (cf. Table 5), the deviation of the approximated conditional probabilities is comparatively small. Although the statistical probability values $P(\omega)$ have been abstracted by qualitative values and the approximation in equation (18) might appear somewhat coarse, the results are nevertheless quite accurate. This illustrates that the qualitative abstraction of the original probabilities conserves enough information to be useful in handling questions of structural relationship.

We have proposed an approach to qualitative knowledge discovery that followed the mechanisms of reverse inductive knowledge representation developed in [5] but is based on a qualitative representation of the empirically obtained probability distribution P that serves as input to the data mining process. An ordinal conditional function κ^P based on qualitative probabilities [1] was used to capture the qualitative information inherent to P. With the use of an algebraic theory of conditionals, the approach generates default rules that are apt to compactly represent the information of κ^P. We briefly described the theoretical and methodological background, and also made clear how our method can be implemented by sketching an algorithm.

A problem of open research is the question, of how to determine the abstraction parameter that is needed to represent the probabilities as polynomials in that parameter in an optimal way. As mentioned before, this determination is crucial when computing the qualitative abstractions of the information inherent in the original distribution because the precision of the computed qualitative representation depends particularly on the chosen parameter.

The purely probabilistic version of the described algorithm has been developed and implemented during the CONDOR-project[3]. CONDOR is an integrated system for learn-

[2] At this point, is does not matter whether the ranking values originating directly from equation (4) or the normalized ones are used, because the normalization constant will be cancelled out when considering conditionals.

[3] Supported by the German Research Society, DFG, under grant BE 1700/5-1.

ing, reasoning and belief revision in a probabilistic environment. For future work, we are planning to implement the algorithm for qualitative knowledge discovery presented in this paper and integrate it into CONDOR to also provide qualitative learning and reasoning facilities. The common methodological grounds based on c-representations which can be used both for probabilistic and default reasoning will establish clear links between quantitative and qualitative frameworks, as was illustrated in the running example of this paper.

7 Summary and Further Work

We have proposed an approach to qualitative knowledge discovery that followed the mechanisms of reverse inductive knowledge representation developed in [5] but is based on a qualitative representation of the empirically obtained probability distribution P that serves as input to the data mining process. An ordinal conditional function κ^P based on qualitative probabilities [1] was used to capture the qualitative information inherent to P. With the use of an algebraic theory of conditionals, the approach generates default rules that are apt to compactly represent the information of κ^P. We briefly described the theoretical and methodological background, and also made clear how our method can be implemented by sketching an algorithm.

A problem of open research is the question, of how to determine the abstraction parameter that is needed to represent the probabilities as polynomials in that parameter in an optimal way. As mentioned before, this determination is crucial when computing the qualitative abstractions of the information inherent in the original distribution because the precision of the computed qualitative representation depends particularly on the chosen parameter.

The purely probabilistic version of the described algorithm has been developed and implemented during the CONDOR-project[4]. CONDOR is an integrated system for learning, reasoning and belief revision in a probabilistic environment. For future work, we are planning to implement the algorithm for qualitative knowledge discovery presented in this paper and integrate it into CONDOR to also provide qualitative learning and reasoning facilities. The common methodological grounds based on c-representations which can be used both for probabilistic and default reasoning will establish clear links between quantitative and qualitative frameworks, as was illustrated in the running example of this paper.

References

1. Goldszmidt, M., Pearl, J.: Qualitative probabilities for default reasoning, belief revision, and causal modeling. Artificial Intelligence (1996)
2. Benferhat, S., Dubois, D., Prade, H.: Nonmonotonic reasoning, conditional objects and possibility theory. Artificial Intelligence (92), 259–276 (1997)
3. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44, 167–207 (1990)

[4] Supported by the German Research Society, DFG, under grant BE 1700/5-1.

4. Kern-Isberner, G.: Solving the inverse representation problem. In: Proceedings 14th European Conference on Artificial Intelligence, ECAI 2000, Berlin, pp. 581–585. IOS Press, Amsterdam (2000)
5. Kern-Isberner, G., Fisseler, J.: Knowledge discovery by reversing inductive knowledge representation. In: Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004, pp. 34–44. AAAI Press, Menlo Park (2004)
6. Adams, E.: Probability and the logic of conditionals. In: Hintikka, J., Suppes, P. (eds.) Aspects of inductive logic, pp. 265–316. North-Holland, Amsterdam (1966)
7. Spohn, W.: Ordinal conditional functions: a dynamic theory of epistemic states. In: Harper, W., Skyrms, B. (eds.) Causation in Decision, Belief Change, and Statistics, vol. 2, pp. 105–134. Kluwer Academic Publishers, Dordrecht (1988)
8. Benferhat, S., Dubois, D., Lagrue, S., Prade, H.: A big-stepped probability approach for discovering default rules. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS) 11, 1–14 (2003)
9. Paris, J.: The uncertain reasoner's companion – A mathematical perspective. Cambridge University Press, Cambridge (1994)
10. Kern-Isberner, G.: Characterizing the principle of minimum cross-entropy within a conditional-logical framework. Artificial Intelligence 98, 169–208 (1998)
11. Kern-Isberner, G.: Conditionals in Nonmonotonic Reasoning and Belief Revision. LNCS (LNAI), vol. 2087. Springer, Heidelberg (2001)
12. Jain, A., Murty, M., Flynn, P.: Data clustering: A review. ACM Computing Surveys 31(3) (1999)
13. Cooper, G., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Machine Learning 9, 309–347 (1992)
14. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction and Search. Lecture Notes in Statistics, vol. 81. Springer, Heidelberg (1993)
15. Heckerman, D.: Bayesian networks for knowledge discovery. In: Fayyad, U., Piatsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) Advances in knowledge discovery and data mining. MIT Press, Cambridge (1996)
16. Buntine, W.: A guide to the literature on learning probabilistic networks from data. IEEE Transactions on Knowledge and Data Engineering 8(2), 195–210 (1996)
17. Herskovits, E., Cooper, G.: Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. Technical Report KSL-90-22, Knowledge Systems Laboratory (1990)
18. Geiger, D.: An entropy-based learning algorithm of bayesian conditional trees. In: Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence, pp. 92–97 (1992)
19. Kern-Isberner, G.: Following conditional structures of knowledge. In: Burgard, W., Christaller, T., Cremers, A.B. (eds.) KI 1999. LNCS (LNAI), vol. 1701, pp. 125–136. Springer, Heidelberg (1999)
20. Kern-Isberner, G.: Handling conditionals adequately in uncertain reasoning. In: Benferhat, S., Besnard, P. (eds.) ECSQARU 2001. LNCS (LNAI), vol. 2143, pp. 604–615. Springer, Heidelberg (2001)
21. DeFinetti, B.: Theory of Probability, vol. 1,2. John Wiley and Sons, New York (1974)
22. Calabrese, P.: Deduction and inference using conditional logic and probability. In: Goodman, I., Gupta, M., Nguyen, H., Rogers, G. (eds.) Conditional Logic in Expert Systems, pp. 71–100. Elsevier, North Holland (1991)
23. Kern-Isberner, G.: A thorough axiomatization of a principle of conditional preservation in belief revision. Annals of Mathematics and Artificial Intelligence 40(1-2), 127–164 (2004)
24. Beierle, C., Kern-Isberner, G.: Modelling conditional knowledge discovery and belief revision by abstract state machines. In: Börger, E., Gargantini, A., Riccobene, E. (eds.) ASM 2003. LNCS, vol. 2589, pp. 186–203. Springer, Heidelberg (2003)

On the Notion of an XML Key

Sven Hartmann[1,⋆], Henning Köhler[2], Sebastian Link[3], Thu Trinh[1], and Jing Wang[4]

[1] Clausthal University of Technology, Germany
sven.hartmann@tu-clausthal.de
[2] University of Queensland, Australia
[3] Victoria University of Wellington, New Zealand
[4] Massey University, New Zealand

Abstract. Ongoing efforts in academia and industry to advance the management of XML data have created an increasing interest in research on integrity constraints for XML. In particular keys have recently gained much attention. Keys help to discover and capture relevant semantics of XML data, and are crucial for developing better methods and tools for storing, querying and manipulating XML data. Various notions of keys have been proposed and investigated over the past few years. Due to the different ways of picking and comparing data items involved, these proposals give rise to constraint classes that differ in their expressive power and tractability of the associated decision problems. This paper provides an overview of XML key proposals that enjoy popularity in the research literature.

1 Introduction

XML, the eXtensible Markup Language [7] has become the standard for sharing and integrating data on the Web and elsewhere. There is wide consensus among researchers and practitioners that XML requires commensurate support by DBMS and data management tools, in particular to store, query and process XML data in its native format. The inherent syntactic flexibility and hierarchical structure of XML make it a challenge to precisely describe desirable properties of XML data and provide methods and facilities that can efficiently conclude, validate, or enforce such properties, cf. [15,16,17,26,28].

Integrity constraints restrict data stores such as XML documents or XML databases to those considered meaningful for some application of interest. Specifying and enforcing integrity constraints helps to ensure that the data stored stays valid and does not deviate from reality. Integrity constraints enjoy a variety of applications ranging from schema design, query optimisation, efficient access and updating, privacy and integrity, data exchange and integration, to data cleaning, cf. [15].

Keys are without any doubt one of the most fundamental classes of integrity constraints. The importance of keys for XML has been recognized by industry

⋆ Corresponding author.

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 103–112, 2008.

and academia. They provide the means for identifying data items in an unambiguous way, an ability that is essential for retrieving and updating data. For relational data, keys are straightforward to define, convenient to use and simple to reason about. For XML data, however the story is more cumbersome due to the particularities of the XML data model. Over the past few years several notions have been proposed and discussed in the research community, including the definitions that have been introduced into XML Schema [27]. While this has established an industry standard for specifying keys, Arenas et al. [2] have shown the computational intractability of the associated consistency problem, that is, the question whether there exists an XML document that conforms to a given DTD or XSD and satisfies the specified keys.

The most popular alternative proposal is due to Buneman et al [10,11] who define keys independently from any schema such as a DTD or XSD. Rather they based keys on the representation of XML data as trees. Keys uniquely identify nodes of interest in such a tree by some selected nodes, their associated complex values or their location.

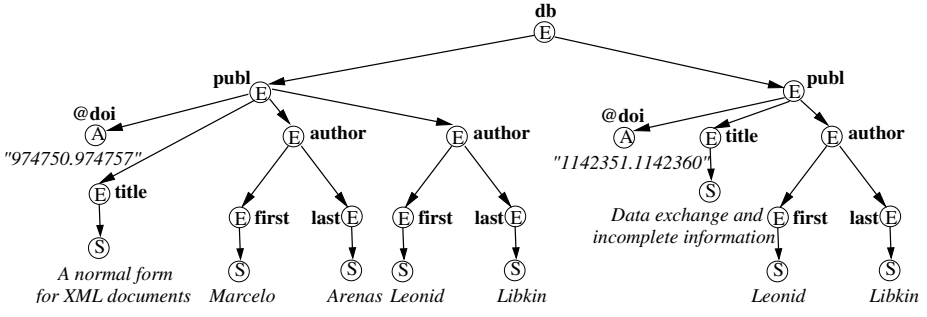


Fig. 1. An XML data tree

In Figure 1, an example of a reasonable key is that the *doi*-value identifies the *publ* node. That is, the *doi* subnodes of different *publ* nodes must have different values. In contrast, an *author* cannot be identified in the entire tree by its *first* and *last* subnodes since the same author can have more than one publication. However, the *author* can indeed be identified by its *first* and *last* subnodes relatively to the *publ* node. That is, for each individual *publ* node, different *author* subnodes must differ on their *first* or *last* subnode value.

Clearly, the expressiveness of such keys depends on the means that are used for selecting nodes in a tree, and by the semantics of the associated node selection queries. The key notion proposed Buneman et al. is flexible in the choice of an appropriate query language. Unfortunately, increased expressive power often comes at the price of increased complexity of the associated decision problems. It is still a major challenge to find natural and useful classes of keys that can be managed efficiently [15,16,17,26,28].

The objective of this paper is to give a brief overview of definitions of keys that have been proposed in the research literature. Our focus is on the key notion of Buneman et al., but we also address alternative proposals [3,4,32].

2 Prerequisites

It is common to represent XML data by ordered, node-labelled trees. Such a representation is e.g. used in DOM [1], XPath [13], XQuery [6], XSL [22], and XML Schema [27]. Figure 1 shows an example of XML data represented as a tree in which nodes are annotated by their type: E for element, A for attribute, and S for text (PCDATA).

In the literature devoted to research on XML constraints, several variations of the tree model for XML data have been used. All of them simplify the XML standard [7] by concentrating on its major aspects. Here we follow the approach taken in [10,11]. We assume that there are three mutually disjoint non-empty sets $\mathbf{E}$, $\mathbf{A}$, and $\mathbf{S} = \{S\}$. In the sequel, $\mathbf{E}$ will be used for element names, $\mathbf{A}$ for attribute names, and S for denoting text. We further assume that these sets are pairwise disjoint, and put $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \mathbf{S}$. We refer to the elements of $\mathcal{L}$ as *labels*.

2.1 XML Trees and Paths

An *XML tree* is defined as a 6-tuple $T = (V, lab, ele, att, val, r)$. Herein V denotes the set of *nodes* of T, and lab is a mapping $V \to \mathcal{L}$ assigning a label to every node in V. The XML tree is said to be *finite* if V is finite. A node v in V is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) \in \mathbf{S}$. Further, ele and att are partial mappings defining the edge relation of T: for any node v in V, if v is an element node, then $ele(v)$ is a list of element and text nodes in V, and $att(v)$ is a set of attribute nodes in V. If v is an attribute or text node then $ele(v)$ and $att(v)$ are undefined. For a node $v \in V$, each node w in $ele(v)$ or $att(v)$ is called a *child* of v, and we say that there is an *edge* (v, w) from v to w in T. Let E_T denote the set of edges of T, and let E_T^* denote the transitive closure of E_T. Further, val is a partial mapping assigning a string value to each attribute and text node: for any node v in V, if v is an attribute or text node then $val(v)$ is a string, and $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished *root node* of T.

A *path expression* is a finite sequence of zero or more symbols from some alphabet $\mathbf{M}$. The unique sequence of zero symbols is called the *empty path expression*, denoted by ε, and a dot (.) is used to denote the concatenation of path expressions. The set of all path expressions over $\mathbf{M}$, with the binary operation of concatenation and the identity ε form a free monoid. We are in particular interested in path expressions over the alphabet $\mathcal{L}$ which we call *simple*.

A simple path p of an XML tree T is a sequence of nodes $v_0, \ldots, v_m$ where (v_{i-1}, v_i) is an edge for $i = 1, \ldots, m$. We call p a simple path from v_0 to v_m, and say that v_m is *reachable* from v_0 following the simple path p. The path p gives rise to a simple path expression $lab(v_1).\cdots.lab(v_m)$, which we denote by $lab(p)$. An XML tree T has a tree structure: for each node v of T, there is a unique simple path from the root r to v.

2.2 Value Equality

Next we define value equality for pairs of nodes in XML trees. Informally, two nodes u and v of an XML tree T are value equal if they have the same label and, in addition, either they have the same string value if they are text or attribute nodes, or their children are pairwise value equal if they are element nodes. More formally, two nodes $u, v \in V$ are *value equal*, denoted by $u =_v v$, if and only if the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. That is, two nodes u and v are value equal when the following conditions are satisfied:

(a) $lab(u) = lab(v)$,
(b) if u, v are attribute or text nodes, then $val(u) = val(v)$,
(c) if u, v are element nodes, then *(i)* if $att(u) = \{a_1, \ldots, a_m\}$, then $att(v) = \{a'_1, \ldots, a'_m\}$ and there is a permutation π on $\{1, \ldots, m\}$ such that $a_i =_v a'_{\pi(i)}$ for $i = 1, \ldots, m$, and *(ii)* if $ele(u) = [u_1, \ldots, u_k]$, then $ele(v) = [v_1, \ldots, v_k]$ and $u_i =_v v_i$ for $i = 1, \ldots, k$.

For example, the second and third *author* node (according to document order) in Figure 1 are value equal. Notice that the notion of value equality takes the document order of the XML tree into account. We remark that $=_v$ defines an equivalence relation on the node set of an XML tree.

Consider two subsets U and W of V. We call U and W *value equal* if there exists a bijection $\beta : U \to W$ such that $u =_v \beta(u)$ for all $u \in U$. The *value intersection* $U \cap_v W$ of U and W consists of all pairs $(u, w) \in U \times W$ such that $u =_v w$ holds, and the *value difference* $U -_v W$ consists of all nodes in U that are not value equal to any node in W.

2.3 Node Selection Queries

A *node selection query* Q defines a mapping $[\![Q]\!]_T : V \to 2^V$ that assigns every node $v \in V$ a subset of V, called the *selected nodes*, that may be seen as the result of executing the query at node v.

For node selection queries we can define operations like union, intersection, concatenation, reverse, absolution and the identity as follows, cf. [13]:

$$
\begin{aligned}
[\![Q_1 \cup Q_2]\!]_T(v) &:= [\![Q_1]\!]_T(v) \cup [\![Q_2]\!]_T(v) \\
[\![Q_1 \cap Q_2]\!]_T(v) &:= [\![Q_1]\!]_T(v) \cap [\![Q_2]\!]_T(v) \\
[\![Q_1.Q_2]\!]_T(v) &:= \{x : w \in [\![Q_1]\!]_T(v), x \in [\![Q_2]\!]_T(w)\} \\
[\![Q^R]\!]_T(v) &:= \{x : v \in [\![Q]\!]_T(x)\} \\
[\![Q^A]\!]_T(v) &:= [\![Q]\!]_T(r) \\
[\![\varepsilon]\!]_T(v) &:= \{v\}
\end{aligned}
$$

Depending on the particular application one aims to identify query languages that enable users to retrieve as much data as possible that is relevant for the

underlying application domain, yet sufficiently simple to process the queries efficiently. A problem that has been widely studied in the literature due to its immediate practical relevance is the containment problem. A node selection query Q is said to be *contained* in a node selection query Q', denoted by $Q \sqsubseteq Q'$, if for every XML tree T and every node $v \in V$ we have that $[\![Q]\!]_T(v)$ is a subset of $[\![Q']\!]_T(v)$. Two queries are (semantically) *equivalent*, denoted by $Q \equiv Q'$, if they contain one another. The *containment problem* for a class of queries asks to decide containment, while the *equivalence problem* asks to decide equivalence for the query class under inspection.

It remains to find a convenient way to express useful node selection queries that can be evaluated efficiently. In the literature regular languages of path expressions have been widely used for this purpose. Alternatively, XPath expressions [13] are of course popular, too. For recent tractability results on the containment and equivalence problem for regular path languages and XPath fragments we refer to [5,14,18,23,24,25,31].

3 Keys for XML

In this section we attempt to give a definition of keys for XML data that is sufficiently general to cover most of the existing proposals for defining such constraints. We should note that this definition might not always provide the most convenient way of expressing the constraint at hand nor reveal the most efficient way for tackling the associated decision problems.

An *XML key* is a pair $\sigma = (C, Q, \mathcal{F})$ where C and Q are node selection queries, and $\mathcal{F} = \{F_1, \ldots, F_k\}$ is a finite set of node selection queries. Adapting the terminology of [27], we call C the *context*, Q the *selector* and $F_1, \ldots, F_k$ the *fields* of the key σ. Given an XML tree T, $[\![C]\!]_T(r)$ is called the *context node* set. For any context node u, $[\![Q]\!]_T(u)$ is called the *target node* set, and for any target node v and every $i = 1, \ldots, k$ we call $[\![F_i]\!]_T(v)$ a *key node* set.

An XML tree T *satisfies* an XML key $\sigma = (C, Q, \mathcal{F})$ if for any target nodes u, v that belong to the same target node set it holds that if for all $i = 1, \ldots, k$ their key node sets $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ *agree* then the nodes u and v themselves *agree*.

3.1 Agreement of Nodes

In the literature different authors have suggested different approaches for defining *agreement* of target nodes and of key node sets. To continue with we will summarize the most popular proposals. For the *agreement* of two key node sets the following criteria are potentially of interest:

(Ka) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are equal,
(Kb) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ have non-empty intersection,
(Kc) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are value equal,
(Kd) $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ have non-empty value intersection.

For the *agreement* of two target nodes u and v the following criteria are potentially of interest:

(Ta) $u = v$, that is, u and v are identical nodes,
(Tb) $u =_v v$, that is, u and v are value equal nodes.

Moreover, one might want to combine these requirements with one or more of the following criteria for some or all $i = 1, \ldots, k$:

(Tc) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ are non-empty,
(Td) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ contain at most one node,
(Te) both $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ contain only attribute or text nodes.

3.2 Strong Keys Defined in XML Schema

As an example we consider keys as defined in XML Schema [27]. For such a key σ to hold it is necessary that for each target node v each of the key fields F_i with $i = 1, \ldots, k$ selects exactly one key node. This prerequisite calls for uniqueness and existence. Buneman et. al [9] call a key with such a prerequisite *strong*.

XML Schema [27] uses criterion (Kd) above for the agreement of key node sets. Note that if the prerequisite holds then criteria (Kc) and (Kd) coincide, and so do criteria (Ka) and (Kb). Moreover, this prerequisite imposes a condition on each target node. Even if there is only a single target node v in an XML tree T the key will be violated when one of the key node sets $[\![F_i]\!]_T(v)$ is not a singleton set. To avoid that in such a case the target node v agrees with itself one chooses criteria (Ta,Tc,Td) for the agreement of target nodes. Further, criterion (Te) accounts for the restriction of value equality to string equality in XML Schema.

XML Schema further defines a weaker version of the previous key notion, called *unique*. For such a constraint σ to hold it is necessary that for each target node v each of the key fields F_i with $i = 1, \ldots, k$ selects no more than one key node. Again criterion (Kd) is used for the agreement of key node sets, and criteria (Ta,Td,Te) for the agreement of target nodes.

3.3 Absolute and Relative Keys

A key of the form $(\varepsilon, Q, \mathcal{F})$ is called *absolute*. That is, in case of an absolute key the root node is the single context node. To emphasize that in general the key context C may also be chosen differently, one speaks of *relative* keys. It should be noted that every relative key may be rewritten as an absolute key by transforming the key context C into an additional key field with criterion (Kb) above: It can be shown that an XML tree satisfies $(C, Q, \mathcal{F})$ if and only if it satisfies $(\varepsilon, C.Q, \mathcal{F} \cup \{C^A \cap Q^R\})$. However the latter representation of the key might be less intuitive in many cases.

4 Popular Existing Proposals for XML Keys

4.1 Keys Defined by Buneman et al.

In [8,10] Buneman et al. introduce and study absolute and relative keys that do not have the uniqueness and existence prerequisite of the strong keys defined in

XML Schema [27]. This proposal is mainly motivated by the observation that strong keys are not always finitely satisfiable. That is, there are keys for which there is no a finite XML tree that satisfy them. To overcome this situation, Buneman et al. define keys using criteria (Kd) for the agreement of key node sets, and only (Ta) for the agreement of target nodes.

For node selection queries they define the path language PE consists of all path expressions over the alphabet $\mathcal{L} \cup \{\_, \_^*\}$, with the binary operation of concatenation and the empty path expression ε as identity. Herein, $\_$ and $\_^*$ are symbols not in $\mathcal{L}$ that serve as the *single symbol wildcard* and the *variable length wildcard*. The semantics of path expressions from PE is defined by putting:

$$\begin{aligned}
[\![\ell]\!]_T(v) &:= \{w \mid (v,w) \in E_T, lab_T(w) = \ell\} \\
[\![_]\!]_T(v) &:= \{w \mid (v,w) \in E_T\} \\
[\![_^*]\!]_T(v) &:= \{w \mid (v,w) \in E_T^*\}
\end{aligned}$$

The semantics of the concatenation operator and of ε is defined as for general node selection queries. When comparing PE and XPath [13], one observes the equivalences $\varepsilon \equiv .$ and $\_ \equiv *$ and $\_^* \equiv .//.$ and $Q_1.Q_2 \equiv Q_1/Q_2$ and $Q_1//Q_2 \equiv Q_1.\_^*.Q_2$ showing that PE corresponds to the XPath fragment $XP(.,/,*,//)$.

In [8,10] PE expressions are used for the context C and the key selector Q, while simple path expressions from are used for the key fields $F_1, \ldots, F_k$. At the end of [10] the use of PE expressions for the key fields is briefly discussed based on a more restrictive definition of value intersection for key node sets: The *limited value intersection* $[\![F_i]\!]_T(u) \cap_{lv} [\![F_i]\!]_T$ of key node sets $[\![F_i]\!]_T(u)$ and $[\![F_i]\!]_T(v)$ consists of all pairs $(x,y) \in [\![F_i]\!]_T(u) \times [\![F_i]\!]_T(v)$ such that $x =_v y$ holds and for which a simple path expression $F \sqsubseteq F_i$ with $x \in [\![F_i]\!]_T(u)$ and $y \in [\![F_i]\!]_T(v)$ exists.

As an example consider the XML tree in Figure 1 and suppose we replace the *author* nodes by *firstauthor* and *secondauthor* nodes. Suppose further we have a key $(\varepsilon, \_^*.publ, \_^*.last)$ with the *publ* nodes as targets. The field $\_^*.last$ would then pick two *last* nodes for the first *publ* node, and one *last* for the second *publ* node. The value intersection of the two key node sets would contain the two *last* nodes for Libkin as a pair, while the limited value intersection would be empty.

In [9,11] Buneman et al. study the axiomatisability and implication problem of the keys introduced in [8,10]. This time they restrict themselves to the path language PL consisting of all path expressions over the alphabet $\mathcal{L} \cup \{\_^*\}$. PL expressions are used for the key context C, the key selector Q and the key fields $F_1, \ldots, F_k$. In [9] agreement of key node sets is is based on limited value intersection, while in [11] it is based on the original definition of value intersection.

Several authors have continued to investigate relevant properties for this class of keys. Without claiming to be complete we will mention a few. In [12] Chen et al. study efficient ways for validating keys against an XML tree. Their procedure can be used for checking the semantic correctness of an XML tree and for checking incremental updates made to an XML tree. Its asymptotic performance is linear in the size of the input tree and the input keys.

In [19] Grahne and Zhu develop efficient methods for mining keys that are satisfied by an XML tree. The keys they search for use path expressions over the alphabet $\mathcal{L} \cup \{\_\}$. Their algorithm also looks for keys that hold only in certain parts of the tree. These approximate keys may serve as candidate keys for views.

In [29] Vincent et al. use absolute keys with simple path expressions as selectors and fields $F_1, \ldots, F_k$ and with $k \geq 1$. In particular, they emphasize that such a key corresponds to a strong functional dependency as defined and studied in [29]. Note that Buneman et al. [11] give an example of a key with $k = 0$ and emphasize that such a key may cause inconsistency in the presence of a DTD.

In [21] Hartmann and Link give an axiomatisation for the class of keys where context C and selector Q are PL expressions, and the fields F_i are simple path expressions with $k \geq 1$. The completeness proof is based on a characterisation of key implication in terms of reachability of nodes in a suitable digraph constructed from a key to reasoned about. Utilising the efficient evaluation of Core XPath queries [18] this gives rise to a decision procedure for the implication of this key class that is quadratic in the size of the input keys.

More recently, Wang [30] has investigated the axiomatisability and implication problem for keys where the path expressions may include the single symbol wildcard _. Her results exploit a correspondence between XML keys and tree patterns that have been used by Miklau and Suciu [23] to study containment problem for certain XPath fragments.

In [20] Hartmann and Link extend XML keys to numerical constraints for XML. While keys are intended to uniquely determine nodes in an XML tree, numerical constraints only do this up to a certain number of nodes. For this number one may specify bounds. The implication problem is shown to be intractable for some classes of numerical constraints. For the class of *numerical keys* with arbitrary integers as upper bounds and with PL expressions as context and selector, simple path expressions as fields, and $k \geq 1$, however, implication can be decided in quadratic time using shortest path methods.

4.2 Keys Defined by Arenas et al.

Motivated by the key notion of XML Schema, Arenas et al. [2,3,4] study absolute and relative keys. The focus is on the investigation of consistency problems for strong keys (and strong foreign keys). As context C and selector Q they use path expressions of the form $\_^*.\ell$ with $\ell \in \mathbf{E}$, and as key fields $F_1, \ldots, F_k$ labels from $\mathbf{A}$, where $k \geq 1$. For the agreement of target nodes they choose criterion (Ta), and for the agreement of key node sets criterion (Kd) for all $i = 1, \ldots, k$. It should be noted that key fields are limited to the labels of attributes whose existence is guaranteed by a DTD (or XSD). At the same time the uniqueness of attributes is guaranteed by the XML standard [7]. Thus, $F_i(v)$ is a singleton set for every $i = 1, \ldots, k$, such that criteria (Tc,Td,Te) for the agreement of target nodes are automatically satisfied.

In [3,4] Arenas et al. further study absolute keys with a selector Q of the form $Q'.\ell$ where $\ell \in \mathbf{E}$ and Q' denotes a regular expression over the alphabet $E \cup \{\_\}$. In [2] the discussion is extended to absolute keys with path expressions as permitted

by XML Schema [27]. In particular, Arenas et al. show that the consistency problem is NP-hard already for strong keys with $k = 1$ in the presence of a non-recursive and no-star DTD.

4.3 Keys Defined by Yu and Jagadish

In [32] Yu and Jagadish study data redundancies in XML that are caused by functional dependencies. The focus is on the development of an efficient partition-based algorithm for discovering certain functional dependencies, including keys.

For that they also give a definition of absolute keys for XML that shares some similarity with the notion of Buneman et al. First of all, [32] does not take document order into account. This leads to a different notion of value equality: Two nodes that are value equal in [32] might not be value equal by the original definition above if the order of their element children differ. For the agreement of target nodes Yu and Jagadish choose criterion (Ta), and criterion (Kc) for the agreement of key node sets for all $i = 1, \ldots, k$. For the key selector they use simple path expressions, and for the key fields expressions from the XPath fragment $XP(., /, ..)$. That is, key fields may involve simple upward steps.

Acknowledgement

We like to thank Bernhard Thalheim for his constant encouragement and support during the preparation of this paper.

References

1. Apparao, V., et al.: Document object model (DOM) level 1 specification, W3C recommendation (October 1998), http://www.w3.org/TR/REC-DOM-Level-1/
2. Arenas, M., Fan, W., Libkin, L.: What's hard about XML schema constraints? In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 269–278. Springer, Heidelberg (2002)
3. Arenas, M., Fan, W., Libkin, L.: Consistency of XML specifications. In: Akiyama, J., Baskoro, E.T., Kano, M. (eds.) IJCCGGT 2003. LNCS, vol. 3330, pp. 15–41. Springer, Heidelberg (2005)
4. Arenas, M., Fan, W., Libkin, L.: On the complexity of verifying consistency of XML specifications. SIAM J. Comput. 38(3), 841–880 (2008)
5. Benedikt, M., Fan, W., Kuper, G.M.: Structural properties of XPath fragments. Theor. Comput. Sci. 336(1), 3–31 (2005)
6. Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML query language, W3C Proposed Recommendation (November 2006), http://www.w3.org/TR/xquery/
7. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (3rd edn.), W3C Recommendation (Febuary 2004), http://www.w3.org/TR/2004/REC-xml-20040204/
8. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. In: WWW, vol. 10 (2001)

9. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. In: DBPL, pp. 133–148 (2001)
10. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. Computer Networks 39(5), 473–487 (2002)
11. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. Information Systems 28(8), 1037–1063 (2003)
12. Chen, Y., Davidson, S.B., Zheng, Y.: XKvalidator: a constraint validator for XML. In: CIKM 2002 (2002)
13. Clark, J., DeRose, S.: XML path language (XPath) version 1.0, W3C Recommendation (November 1999), http://www.w3.org/TR/xpath
14. Deutsch, A., Tannen, V.: Containment and integrity constraints for XPath. In: KRDB (2001)
15. Fan, W.: XML constraints. In: DEXA Workshops, pp. 805–809 (2005)
16. Fan, W., Libkin, L.: On XML integrity constraints in the presence of DTDs. J. ACM 49(3), 368–406 (2002)
17. Fan, W., Siméon, J.: Integrity constraints for XML. J. Comput. Syst. Sci. 66(1), 254–291 (2003)
18. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. 30(2), 444–491 (2005)
19. Grahne, G., Zhu, J.: Discovering approximate keys in XML data. In: CIKM 2002: Proceedings of the eleventh international conference on Information and knowledge management, pp. 453–460. ACM, New York (2002)
20. Hartmann, S., Link, S.: Numerical constraints for XML. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 203–217. Springer, Heidelberg (2007)
21. Hartmann, S., Link, S.: Unlocking keys for XML trees. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 104–118. Springer, Heidelberg (2006)
22. Kay, M.: XSL transformations (XSLT) version 2.0 W3C Candidate Recommendation (November 2005), http://www.w3.org/TR/xslt20/
23. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)
24. Neven, F., Schwentick, T.: XPath containment in the presence of disjunction, DTDs, and variables. Logical Methods in Computer Science 2(3) (2006)
25. Schwentick, T.: XPath query containment. SIGMOD Record 33(1), 101–109 (2004)
26. Suciu, D.: On database theory and XML. SIGMOD Record 30(3), 39–45 (2001)
27. Thompson, H., Beech, D., Maloney, M., and Mendelsohn, N.: XML Schema part 1: Structures second edition, W3C Recommendation, (October 2004), http://www.w3.org/TR/xmlschema-1/
28. Vianu, V.: A web odyssey. SIGMOD Record 32, 68–77 (2003)
29. Vincent, M., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. ACM ToDS 29(3), 445–462 (2004)
30. Wang, J.: Using tree patterns for flexible handling of XML keys. Master's thesis, Massey University (2008)
31. Wood, P.T.: Containment for XPath fragments under DTD constraints. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 297–311. Springer, Heidelberg (2002)
32. Yu, C., Jagadish, H.V.: Efficient discovery of XML data redundancies. In: VLDB, pp. 103–114 (2006)

Embodied Context Semantics

Ander Altuna

School of Computing, University of Leeds, Leeds, UK
Infotech Unit, Robotiker-Tecnalia, Zamudio, Spain
ander@comp.leeds.ac.uk

Abstract. It is a desirable feature of knowledge bases that they are able to accommodate and reason across the different perspectives that may exist on a particular theory or situation. With the aim of obtaining an adequate logic for this problem, the knowledge representation community has extensively researched into the formalization of contexts as first-class citizens. However, most of the proposed logics of context only deal with the propositional case, which for many applications is not enough, and those tackling the quantificational case face many counterintuitive restrictions. In this paper, we present a model-theoretic semantics that, based on a cognitive approach to the notions of context and meaning, succeeds in addressing the quantificational case in a flexible manner that overcomes the limitations of the previous initiatives. The expressive power of the system will be evaluated in the paper by formalizing some of the benchmark examples that can be found in the literature.

1 Introduction

In Knowledge Bases logic is mainly used to infer the consequences and to check the consistency of formally described theories. The predicate calculus and the decidable fragments of it, such as some Description Logics [22], are by far the most commonly adopted logics for the representation of knowledge. However, the language of first-order logic was originally designed to formalize the foundations of mathematics, in which the notion of context is irrelevant, and not our ordinary theories which are inherently subject to a particular perspective. Therefore, although the semantics of the predicate calculus has proved to be very successful in the representation of what Quine [23] came to call the "eternal truths" of mathematics, its application to the theories we normally aim to develop does not result adequate, since their scope is not so broad and it is always relatively easy to find a more general context in which a particular theory fails to hold.

With the aim of solving this problem, to which McCarthy referred [19] as that of Generality in Artificial Intelligence, a new research programme emerged with the intention of creating a logic of context [20] whose language would be capable of referring to contexts as first-class individuals, so that the scope of the expressed theories could be delimited. However, the ultimate goal of the logic was not limited to accommodate the different perspectives that may exist on a particular theory but also looked into enabling the reasoning across different

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 113–125, 2008.

contexts, so that the facts that hold in a given context can affect the ones that hold in another.

Since this programme was initiated, many logics have been proposed to formalize contexts, among which the best known are [4] ,[5], [9], and [13]. However, most of these logics only deal with the propositional case which, as shown by the benchmark examples included in [12], is not expressive enough in most of the cases. And those that address the quantificational case [4] face many counterintuitive restrictions, like the imposition of constant domains or the requirement that the same facts hold in a particular context regardless of the outer context in which it is described.

Although the cognitive approach to contexts of these logics has helped in overcoming many of the problems of the approaches that, in the metaphysical tradition outside the AI community, considered contexts as part of the structure of the world ([15], [17]), we think that a satisfactory logic of context that aims to model all the desiderata of context will need to go one step further by defining the notion of meaning from a cognitive perspective. We judge that it is well justified by recent findings in cognitive science ([16], [8]) to consider meaning as a cognitive function and not as an abstract metaphysical notion. In our view, the externalist semantics of the previous initiatives, in which meaning is considered to be a disembodied function, is the cause of their problems and limitations.

In this paper we present a logic of context that, based on a cognitive approach to both context and meaning, succeeds in addressing the quantificational case in a flexible manner that overcomes the problems and limitations of the previous attempts to build a satisfactory logic of context. However, the importance of the results is increased by the fact that the expressive power of the logic here presented is superior to that of any prior logic of context, since our cognitive approach to meaning allows that not only contexts but also interpretations are treated as first-class citizens.

The paper is structured as follows. In the second section, we will review the AI research on contexts within the framework of knowledge representation. In the third section, we will firstly introduce the intuitions that guide our semantics and the desiderata of a logic of context, and then we will proceed to present the model-theoretic semantics that we propose. In the fourth section, we will show the expressive power of the formal language associated with the semantics by formalizing some of the benchmark examples that can be found in [12]. In the fifth section, we will describe the related work. We will finish the paper by extracting some conclusions.

2 AI Research in Contexts

Although the problem of contexts has been tackled in other different disciplines like cognitive science [7], philosophy [17] and linguistics [15], the scope of this review will be limited to the work done in the area of knowledge representation and reasoning[1].

[1] A more interdisciplinary survey of the notion of context can be found in [14].

We will focus our attention on the Propositional Logic of Context(PLC) [5], the Local Model Semantics/Multi-Context Systems(LMS/MCS) ([9], [11]) and the Quantificational Logic of Context(QLC), which are the best known logics of context[2]. Although the first two systems only deal with the propositional case, the main purpose of this review is to introduce their different intuitions and limitations in the project of the formalization of contexts[3]. This will facilitate the understanding of the comparison of these logics with the semantics we propose in the next sections.

2.1 Propositional Logic of Context

The language of PLC [5] is that of a multi-modal logic whose modalities are constructed in a predicative fashion by means of the expression *ist* and a set of context labels $\mathbb{K}$. So that if k is a context label contained in $\mathbb{K}$ and φ is a proposition, then $ist\ (k, \varphi)$ is a well-formed formula indicating that φ holds in the modality determined by the context k. The nesting of modalities in a formula like $ist\ (k_1, ist(k_2, \varphi))$ amounts to stating that φ is true in the context label resulting from the concatenation of k_1 and k_2. A formula is said to be true in a particular context label if it is satisfied by all the partial assignments associated with that context label. The notion of partial assignment is intended to model the intuition that some propositions contained in an initial global vocabulary may not be part of the local vocabulary of a context and, therefore, the assignments associated with that context will not define a truth value for these propositions. The reasoning across contexts is performed by the so-called *lifting rules*, which are a set of clauses normally in the form $ist\ (k_1, \varphi_1), \ldots, ist\ (k_n, \varphi_n) \supset ist\ (k_{n+1}, \varphi_{n+1})$ that relate the facts that hold in a context with those that hold in another.

Apart from having an expressive power limited to the propositional case and not handling contexts as truly first-class citizens, the notion of local vocabulary that PLC proposes seems to be too restrictive and too weak at the same time.

On one hand, it is too restrictive because it imposes that the description of the facts that hold in an inner context is exclusively done in terms of the local vocabulary of that inner context. However, in many cases this restriction seems certainly counterintuitive. Let us consider the case of an English/Basque speaker that tells us about the beliefs of a Basque friend of hers who does not speak English at all, there is no good reason why she could not describe that context to us by using her English vocabulary, if this is rich enough to cover all the ideas included in her Basque friend's beliefs.

On the other hand, the notion of local vocabulary is too weak because the local vocabulary of a context is obtained by restricting a global vocabulary at the moment of defining the partial assignments associated with that context. In our opinion, the assumption of a global vocabulary contradicts the idea of a local language, we judge that it would be better to talk about a global set of non-logical symbols or alphabet whose types are locally determined. We will describe this idea in more detail in the next section.

[2] A more detailed review of these and other approaches can be found in ([1], [3]).
[3] We refer to [24] for an excellent comparison of their technical details.

2.2 Local Model Semantics/Multi-Context Systems

LMS/MCS ([9], [11]) is based on the concepts of *locality* and *compatibility*. The first one encodes the intuition that the reasoning is local to each context and therefore each context has its own vocabulary and its own set of inference rules. The second one means that, since contexts are assumed to be different perspectives on the same world, there must be a compatibility relation that determines what perspectives can coexist harmoniously and what perspectives are contradictory in case they coexist. This compatibility relation is syntactically expressed through a set of so-called *bridge rules* that, in contrast with the inference rules, does not belong to any context.

Again LMS/MCS does not address the quantificational case and, as we will show in the fifth section, this is not enough for many of the cases. Even in the case that the contextualized formulas are well-formed formulas of the predicate calculus [10], contexts are not treated as first-order citizens and this is a hard limitation in many applications.

On the other hand, the principle of locality in LMS/MCS is so strong that, it does not allow even to refer to the facts that hold in other contexts, if not in the way of artificially constructed auxiliary propositions. We consider that this restriction is too strong since even if the meaning of some symbols is unknown in a certain context these symbols can be used to describe a context and refer to an external interpretation in order to extract their meaning.

Let us consider a situation in which an English/Basque speaker and a monolingual English friend of hers are involved. This time the multilingual speaker is so angry with her English friend that she has promised that for a day she will always say the truth but she will only do it in Basque. After having asked her the time, the English friend could confidently tell us that in the context of the beliefs of her multilingual friend the time is "ordu bata hogeita bost gutxi" when this is interpreted in Basque. In many cases the use of an external vocabulary will be justified by a desire to externalize to a different interpretation the meaning given to some of the symbols contained in a formula, which is a tool heavily used by journalists for example. In the next section, we will show how to materialize this feature.

In our opinion, the fact that LMS/MCS requires the bridge rules not to belong to any particular context contradicts the initial motivations of a logic of context. We think that like any other theory the relations between contexts are subject to different perspectives and therefore it seems reasonable that their scope is also limited.

2.3 Quantificational Logic of Context

In line with the notion of local validity proposed by PLC, the Quantificational Logic of Context [4] determines the truth of a formula in a particular context by associating a set of typed first-order structures with each context. These structures are typed because, in contrast with PLC and LMS/MCS, QLC formalizes contexts as first-class citizens having their own sort. Therefore, QLC enables the ascription of properties and relations to contexts and the quantification over them.

However, QLC requires many counterintuitive restrictions that limit very much its utility. First, it requires that the universe of discourse is the same for every context. It is desirable for a logic of context to allow for flexible domains so that the set of existing individuals can be different depending on the context. Actually, we go one step further and, in the spirit of Modal Realism [18], we identify a context as the sum of its existing individuals, what will be technically detailed in the third section. Second, it forces every constant symbol to denote the same object regardless of the context in which it is interpreted, or in other words, constants are rigid designators. Clearly, this contradicts all what has been previously said about contextual vocabularies. Third, the set of facts that hold in a context is not subject to any particular perspective and, therefore, the way in which contexts are nested is irrelevant. Like in the case of LMS/MCS *bridge rules*, we think that this condition, known as *flatness*, clashes with the very spirit of a logic of context whose original goal is the formalization of every theory as a particular perspective on the world.

3 Embodied Context Semantics

3.1 The Cognitive Approach to Context and Meaning

All these limitations and problems of the prior logics of context ledGuha and McCarthy [12] to elaborate a classification, centred around the concept of *lifting rule*, of the different kinds of situations in which the reasoning across contexts is fundamental. However, we think that the success in the search for a satisfactory logic of context will not only come from a good classification of contexts, but a reconsideration of the notion of meaning used by these logics is also necessary.

In contrast with the metaphysical tradition ([15], [18]) that understands contexts as part of the structure of the world and, therefore, faces the problem of determining the complete set of indexicals that determine a unique context, the cognitive approach to contexts of [4], [5], [9] and [13] considers contexts as the partial information that an agent has about the circumstances in which she performs an interpretation. In our opinion, although the cognitive approach to contexts has been very useful in avoiding the problem with indexicals of the metaphysical tradition, there exists a confusion between the context in which a sentence is interpreted and the context that is being described from which the sentence obtains its referents. In a very different approach to the one presented here, Barwise and Perry [2] came to classify respectively these kinds of contexts as the *situation of utterance*, *focus situation* and the *resource situation* [6].

In most of the cases that [4], [5], [9] and [13] refer to contexts, they are actually regarding them to be more proximate to the concept of described context than to that of context of interpretation. When a context is said to determine its local vocabulary in these logics, it does not seem that this is thought to be due to a shift in the interpretation of the non-logical symbols but to a change in the ground of referents from which the non-logical symbols obtain their denotation. Furthermore, even assuming that both the concept of described context and the concept of context of interpretation are contemplated in [4], [5], [9] and [13],

there is no reason why both roles should always be realized by the same context, which is always the case in these logics.

In our view, the problem is that these logics keep the externalist semantics of the predicate calculus, in which meaning is regarded as an abstract function from symbols to their extension in the world, but adapting their extensions to the case of partial contexts instead of a complete world. Although the externalist notion of meaning has been very successful in the interpretation of axioms expressed in an commonly agreed vocabulary and whose validity is applicable to every conceivable context, we think that a disembodied approach will never be able to model all the desiderata of a logic of context. Its consideration of meaning as a metaphysical function instead of as a situated action excludes the very idea of context.

We claim that, in order to overcome the problems and limitations of the previous logics of context, it is necessary to go one step further and create a *logic of context and interpretation* that, based on a cognitive approach to context and meaning, not only formalizes contexts but also interpretations as first-class citizens. In the next subsection, we present the technical details of the model-theoretic semantics we have developed with this aim.

3.2 The Semantics

Our logic is developed on the intuition that its non-logical symbols do not have a rigid associated type but their typing depends on the interpretation under which they are being considered. Therefore, the alphabet does not contain different sets of typed non-logical symbols but only a set of primitive non-typed non-logical symbols. Although traditionally the variables are not considered as part of the non-logical symbols, we think that there is no good reason why their typing could not be also subject to interpretation and consequently we have neither included a set of variable symbols in the alphabet. The alphabet of the *logic of context and interpretation* consists, therefore, of the following symbols:

1. An enumerable non-empty set *NLS* of non-logical symbols
2. Identity and part relations: $=$, $\leqslant$
3. External and internal negation: $\neg$, $^{-}$
4. Connectives: $\vee$, $\wedge$, $\supset$
5. Quantifiers: $\forall$, $\exists$
6. Alethic modalities: $\Diamond$, $\Box$
7. Auxiliary symbols: : , [,] , “, ”

As we said in the previous subsection, the logic needs to refer to the interpretation that must be used to extract the meaning of a non-logical symbol. Below, we define the set on non-logical expressions *NLEXP* , which includes the non-logical symbols and their quotation under another non-logical symbol. The non-logical expression “ξ”$_{\mu}$ can be read as “ξ when interpreted under μ”.

Definition 1. *NLEXP is the smallest set of expressions X satisfying the following properties:*

1. $\xi \in NLS \Rightarrow \xi \in X$
2. $\xi \in NLS$ *and* $\mu \in X \Rightarrow$ "ξ"$_\mu \in X$

As the typification of the non-logical symbols depends on the interpretation under which are considered, the set of well-formed formulas depends on the model in question. However, we can inductively define the set of *possibly well-formed formulas* that defines the formulas that under a possible interpretation can be well-formed formulas and excludes those that are grammatically incorrect. We think that the notion of possibly well-formed formula is very important in the conceptual separation between the vocabulary and the grammar. While the vocabulary is local and therefore interpretation-sensitive, the grammar is global and therefore interpretation-independent.

Many of the possible well-formed formulas (pwffs) have a similar form to those of first-order modal logic. However, there are some novel symbols whose character will be better understood by showing their reading. First, the logical symbol $\leqslant$ can be read as "is part of". Its treatment as a logical symbol entails that its axiomatization will be part of the axioms of the logic. Second, while the external negation $\neg P(t_1, \ldots, t_n)$ is read as "it is not the case that P applies to $t_1, \ldots, t_n$", the internal negation $\overline{P}(t_1, \ldots, t_n)$ is read as "it is the case that non-P applies to $t_1, \ldots, t_n$", what intuitively demands the existence of $t_1, \ldots, t_n$. Third, the pwff $k : [A]$ facilitates the contextualization of other pwffs and can be read as "it is the case that A holds in k".

Definition 2. *PWFF is the smallest set of expressions X satisfying the following properties:*

1. $t_1, t_2 \in NLEXP \Rightarrow t_1 \circ t_2 \in X$, *where* $\circ \in \{=, \leqslant\}$,
2. $t_1, \ldots, t_n \in NLEXP$ *and* $P \in NLEXP \Rightarrow P(t_1, \ldots, t_n), \overline{P}(t_1, \ldots, t_n) \in X$,
3. $A \in X \Rightarrow \neg A \in X$,
4. $A, B \in X \Rightarrow [A \circ B] \in X$, *where* $\circ \in \{\vee, \wedge, \supset\}$,
5. $x \in NLEXP$ *and* $A \in X \Rightarrow (\forall x)\,[A], (\exists x)\,[A] \in X$,
6. $A \in X \Rightarrow \Diamond A, \Box A \in X$,
7. $k \in NLEXP$ *and* $A \in X \Rightarrow k : [A] \in X$.

Our model is a typed structure based on three different sets, namely $\mathcal{I}_d$, $\mathcal{I}_k$ and $\mathcal{I}_i$, which intuitively contain the objects of the discourse sort, which traditionally form the universe of discourse in first-order structures, the objects of the context sort and the objects of the interpretation sort respectively. The partial ordering $\preccurlyeq$ encodes the mereological[4] intuition that the set of existing objects in a context is equivalent to the sum of its parts. The local vocabulary[5] and the typification of the non-logical symbols under a given interpretation is determined by the language function l. The meaning function m assigns to each interpretation a relation between the non-logical symbols of its language and their set of possible extensions of the corresponding sort. The function r denotes the realization of any member of $\mathcal{I}_d$, $\mathcal{I}_k$ and $\mathcal{I}_i$ as a context in $\mathcal{I}_k$. Finally, ω_k and ω_i are respectively the actual context and the actual interpretation.

[4] For an excellent introduction to mereology see [25].

[5] Note that for simplicity of the presentation, our logic has no functions.

Definition 3. *A model, $\mathfrak{M}$, is a structure $\mathfrak{M} = \langle \mathcal{I}_d, \mathcal{I}_k, \mathcal{I}_i, \preccurlyeq, l, m, r, \omega_k, \omega_i \rangle$ where:*

1. *$\mathcal{I}_d$, $\mathcal{I}_k$ and $\mathcal{I}_i$ are three non-empty sets,*
2. *$\preccurlyeq$ is a partial ordering on $\mathcal{I}$, where $\mathcal{I} = \mathcal{I}_d \cup \mathcal{I}_k \cup \mathcal{I}_i$,*
3. *l is a function $l : \mathcal{I}_i \to \mathcal{L}$, where $\mathcal{L}$ is the collection of structures consisting of the disjoint sets $\mathcal{V}_\sigma, \mathcal{C}_\sigma, \mathcal{P}_{\sigma_1 \times \cdots \times \sigma_n}$ for any $\sigma, \sigma_1, \ldots, \sigma_n \in \{d, k, i\}$,*
4. *m is a function such that*

$$m(\iota) \subseteq \left((\mathcal{C}^\iota_\sigma \times \mathcal{I}_\sigma) \cup \left(\mathcal{P}^\iota_{\sigma_1 \times \cdots \times \sigma_n} \times \mathcal{I}_{\sigma_1} \times \cdots \times \mathcal{I}_{\sigma_n} \right) \right), \tag{3.1}$$

   where ι is a member of $\mathcal{I}_i$ and $\mathcal{C}^\iota_\sigma$ and $\mathcal{P}^\iota_{\sigma_1 \times \cdots \times \sigma_n}$ refer to the vocabulary of ι according to l,
5. *r is a function $r : \mathcal{I} \to \mathcal{I}_k$,*
6. *ω_k is a member of $\mathcal{I}_k$ and ω_i is a member of $\mathcal{I}_i$.*

Let $\mathfrak{M}$ be a model such that $\mathfrak{M} = \langle \mathcal{I}_d, \mathcal{I}_k, \mathcal{I}_i, \preccurlyeq, l, r, m, \omega_k, \omega_i \rangle$, ι is a member of $\mathcal{I}_i$ and $l(\iota) = \langle , \mathcal{V}_\sigma, \mathcal{C}_\sigma \mathcal{P}_{\sigma_1 \times \cdots \times \sigma_n} \rangle$, we will use $\mathcal{I}^{\mathfrak{M}}_d, \mathcal{I}^{\mathfrak{M}}_k, \mathcal{I}^{\mathfrak{M}}_i$ and $\mathcal{I}^{\mathfrak{M}}$ to refer to $\mathcal{I}_d, \mathcal{I}_k, \mathcal{I}_i$ and $\mathcal{I}$ respectively; $\mathcal{V}^{\mathfrak{M},\iota}_\sigma, \mathcal{C}^{\mathfrak{M},\iota}_\sigma$ and $\mathcal{P}^{\mathfrak{M},\iota}_{\sigma_1 \times \cdots \times \sigma_n}$ to refer to $\mathcal{V}^\iota_\sigma, \mathcal{C}^\iota_\sigma$ and $\mathcal{P}^\iota_{\sigma_1 \times \cdots \times \sigma_n}$ respectively; $\preccurlyeq^{\mathfrak{M}}$, $m^{\mathfrak{M}}$, $r^{\mathfrak{M}}$, $\omega^{\mathfrak{M}}_k$ and $oi^{\mathfrak{M}}$ to refer to $\preccurlyeq$, m, r, ω_k and ω_i respectively.

Below, we proceed to define the concept of assignment and x-variant assignment that will later facilitate the valuation of the pwffs.

Definition 4. *An assignment φ into a model $\mathfrak{M}$ is a function such that*

$$\varphi(\iota) \subseteq \left(\mathcal{V}^{\mathfrak{M},\iota}_\sigma \times \mathcal{I}^{\mathfrak{M}}_\sigma \right), \tag{3.2}$$

where ι is a member of $\mathcal{I}^{\mathfrak{M}}_i$ and $\sigma \in \{d, k, i\}$.

Definition 5. *Let φ and ψ be two assignments into $\mathfrak{M}$. We say that ψ is an x-variant of φ and therefore belongs to the set x-VARIANT$_\varphi$ if:*

$$\begin{aligned} (\forall \iota, \xi)[\iota \in \mathcal{I}^{\mathfrak{M}}_i \wedge \xi \in \left(\left(\mathcal{A}^{\mathfrak{M},\iota} \cup \mathcal{V}^{\mathfrak{M},\iota}_\sigma \right) \setminus \{x\} \right) \\ \supset \{z \mid \langle \xi, z \rangle \in \varphi(\iota)\} = \{z \mid \langle \xi, z \rangle \in \psi(\iota)\}] \end{aligned} \tag{3.3}$$

Below, we define the valuation of a non-logical symbol under a given interpretation, which is independent of the context that is being described.

Definition 6. *Let ξ be a member of NLS , let $\mathfrak{M}$ be a model whose meaning function is denoted by $m^{\mathfrak{M}}$, let ι be a member of $\mathcal{I}^{\mathfrak{M}}_i$, let φ be an assignment into $\mathfrak{M}$, let $\sigma_1, \ldots, \sigma_n, \sigma \in d, k, i$. The valuation function into $\mathfrak{M}$ under ι and φ, formally $v^{\mathfrak{M},\iota}_\varphi : NLS \to (I^{\mathfrak{M}})^n$, where $n \geqslant 1$, is defined as follows:*

$$\xi \in \mathcal{V}^{\mathfrak{M},\iota}_\sigma \Rightarrow v^{\mathfrak{M},\iota}_{\varphi,\sigma}(\xi) = \left\{ x \in \mathcal{I}^{\mathfrak{M}}_\sigma \mid \langle \xi, x \rangle \in \varphi(\iota) \right\} \tag{3.4}$$

$$\xi \in \mathcal{C}^{\mathfrak{M},\iota}_\sigma \Rightarrow v^{\mathfrak{M},\iota}_{\varphi,\sigma}(\xi) = \left\{ x \in \mathcal{I}^{\mathfrak{M}}_\sigma \mid \langle \xi, x \rangle \in m^{\mathfrak{M}}(\iota) \right\} \tag{3.5}$$

$$\xi \in \mathcal{P}^{\mathfrak{M},\iota}_{\sigma_1 \times \cdots \times \sigma_n} \Rightarrow v^{\mathfrak{M},\iota}_{\varphi,\sigma_1 \times \cdots \times \sigma_n}(\xi) = \left\{ x \in \mathcal{I}^{\mathfrak{M}}_{\sigma_1 \times \cdots \times \sigma_n} \mid \langle \xi, x \rangle \in \varphi(\iota) \right\} \tag{3.6}$$

$$\text{Otherwise, } v^{\mathfrak{M},\iota}_{\varphi,\sigma}(\xi) = \emptyset \tag{3.7}$$

Once we have obtained the valuation of the non-logical symbols, we can proceed to obtain the denotations of the set of non-logical expression in a particular described context.

Definition 7. *Let ξ and μ be two members of NLEXP , let $\mathfrak{M}$ be a model, let ι be a member of $\mathcal{I}_i^{\mathfrak{M}}$, let κ be a member of $\mathcal{I}_k^{\mathfrak{M}}$, let φ be an assignment into $\mathfrak{M}$, let $\sigma_1, \ldots, \sigma_n \in \{d, k, i\}$. The denotation function d into $\mathfrak{M}$ under ι, κ and φ for a sort $\sigma_1 \times \cdots \times \sigma_n$, formally $d^{\mathfrak{M},\iota,\kappa}_{\varphi,\sigma_1\times\cdots\times\sigma_n} : NLEXP \rightarrow \mathcal{I}^{\mathfrak{M}}_{\sigma_1} \times \cdots \times \mathcal{I}^{\mathfrak{M}}_{\sigma_n}$, is inductively defined on the construction of the members of NLEXP as follows:*

$$d^{\mathfrak{M},\iota,\kappa}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(\xi) = \{x \in v^{\mathfrak{M},\iota}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(\xi) \mid x \preccurlyeq^{\mathfrak{M}} \kappa\} \tag{3.8}$$

$$d^{\mathfrak{M},\iota,\kappa}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(\text{“}\xi\text{”}\,\mu) = \bigcup_{\lambda\in d^{\mathfrak{M},\iota,\kappa}_{\varphi,i}(\mu)} d^{\mathfrak{M},\lambda,\kappa}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(\xi) \tag{3.9}$$

Once we have obtained the denotations of the non-logical expressions in every possible described context, we can proceed to define the valuation function of the pwffs.

Definition 8. *Let $\mathfrak{M}$ be a model, j be a member of $\mathcal{I}_i^{\mathfrak{M}}$ and φ an assignment into $\mathfrak{M}$. The valuation of the members of PWFF into $\mathfrak{M}$ under ι and φ is inductively defined on their construction as follows:*

$$\begin{aligned} v^{\mathfrak{M},\iota}_{\varphi}(P(t_1,\ldots,t_n)) = \{x \in \mathcal{I}^{\mathfrak{M}} \mid (\exists y_1,\sigma_1,\ldots,y_n,\sigma_n)\,[y_1 \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_1}(t_1) \\ \wedge \cdots \wedge y_n \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_n}(t_n) \wedge \sigma_1,\ldots,\sigma_n \in \{d,k,i\} \\ \wedge \langle y_1,\ldots,y_n\rangle \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(P)]\} \end{aligned} \tag{3.10}$$

$$\begin{aligned} v^{\mathfrak{M},\iota}_{\varphi}(\overline{P}(t_1,\ldots,t_n)) = \{x \in \mathcal{I}^{\mathfrak{M}} \mid (\exists y_1,\sigma_1,\ldots,y_n,\sigma_n)\,[y_1 \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_1}(t_1) \\ \wedge \cdots \wedge y_n \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_n}(t_n) \wedge \sigma_1,\ldots,\sigma_n \in \{d,k,i\} \\ \wedge \langle y_1,\ldots,y_n\rangle \in (\mathcal{I}_{\sigma_1}\times\cdots\times\mathcal{I}_{\sigma_n}) \setminus d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_1\times\cdots\times\sigma_n}(P)]\} \end{aligned} \tag{3.11}$$

$$\begin{aligned} v^{\mathfrak{M},\iota}_{\varphi}(t_1 \leqslant t_2) = \{x \in \mathcal{I}^{\mathfrak{M}} \mid (\exists y_1,\sigma_1,y_2,\sigma_2)\,[y_1 \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_1}(t_1) \\ \wedge y_2 \in d^{\mathfrak{M},\iota,x}_{\varphi,\sigma_2}(t_2) \wedge \sigma_1,\sigma_2 \in \{d,k,i\} \wedge y_1 \preccurlyeq^{\mathfrak{M}} y_2]\} \end{aligned} \tag{3.12}$$

$$v^{\mathfrak{M},\iota}_{\varphi}(\neg A) = \mathcal{I}^{\mathfrak{M}} \setminus v^{\mathfrak{M},\iota}_{\varphi}(A) \tag{3.13}$$

$$v^{\mathfrak{M},\iota}_{\varphi}([A \vee B]) = v^{\mathfrak{M},\iota}_{\varphi}(A) \cup v^{\mathfrak{M},\iota}_{\varphi}(B) \tag{3.14}$$

$$v^{\mathfrak{M},\iota}_{\varphi}((\exists x)\,[A]) = \bigcup_{\psi\in x\text{-}VARIANT_{\varphi}} v^{\mathfrak{M},\iota}_{\psi}(A) \tag{3.15}$$

$$v^{\mathfrak{M},\iota}_{\varphi}(\Diamond A) = \begin{cases} \emptyset \text{ if } v^{\mathfrak{M},\iota}_{\varphi}(A) = \emptyset \\ I^{\mathfrak{M}} \text{ if not } v^{\mathfrak{M},\iota}_{\varphi}(A) = \emptyset \end{cases} \tag{3.16}$$

$$\begin{aligned} v^{\mathfrak{M},\iota}_{\varphi}(t:[A]) = \{x \in \mathcal{I}^{\mathfrak{M}} \mid (\exists y,z)[y \in d^{\mathfrak{M},\iota,x}_{\varphi,k}(t) \wedge y = r^{\mathfrak{M}}(z) \\ \wedge z \in v^{\mathfrak{M},\iota}_{\varphi}(A)]\} \end{aligned} \tag{3.17}$$

From the valuation of the pwffs, we can determine whether a formula is supported or not by a model.

Definition 9. *Let $\mathfrak{M}$ be a model, let φ be an assignment into $\mathfrak{M}$ and let A be in PWFF . We write $\mathfrak{M} \Vdash_\varphi A$ iff $\omega_k^{\mathfrak{M}} \in v_\varphi^{\mathfrak{M},\omega_i^{\mathfrak{M}}}(A)$, where $\Vdash$ is pronounced as "supports".*

Finally, we proceed to define the notions of satisfiability and validity.

Definition 10. *Let $\mathfrak{M}$ be a model, let φ be an assignment into $\mathfrak{M}$ and let A be in PWFF . We say that A is:*

1. *Satisfiable in $\mathfrak{M}$ iff there is an assignment φ such that $\mathfrak{M} \Vdash_\varphi A$.*
2. *Satisfiable iff there is a model $\mathfrak{M}$ in which A is satisfiable.*
3. *Valid in $\mathfrak{M}$ iff for every assignment φ, $\mathfrak{M} \Vdash_\varphi A$.*
4. *Valid iff A is valid in every model.*
5. *Unsatisfiable iff there is no model in which A is satisfiable.*

4 Some Illustrative Examples

In order to show the expressive power of the *logic of context and interpretation* and give a flavour of the applications for which it can be used, we will formalize some of the benchmark examples suggested in the classification of contexts that can be found in [12]. Although we do not claim that this classification of contexts is definitive, we think that the examples it contains, some of which cannot be formalized in the previous logics of context, pose a big challenge for a logic that claims to formalize context in a satisfactory way. The classification divides the contexts based on their assumption and simplifications and according to the kind of *lifting rules* they require. In this sense, it concludes that four major groups of contexts exist, namely *projection contexts*, *approximation contexts*, *ambiguity contexts* and *mental state contexts*. Below, we formalize three examples that exploit much of the expressive power of our logic.

Example 1. A classical example is that of the normalcy/kindness conditions, which falls into the category of the projection contexts. Below is shown an instance of this example stating that every travel context is developed under normal conditions. As can be seen, our logic is capable of quantifying over contexts and therefore there is no problem in the formalization of this example.

$$(\forall k)[\text{NormalConditions}(k) \supset k : [(\forall x)[\text{clothed(x)} \wedge \text{conscious(x)} \wedge \ldots]]] \tag{4.1}$$

$$(\forall k)[\text{TravelContext}(k) \supset k : [(\forall x)[\text{haveTicket}(x) \wedge \text{atAirport}(x) \supset \text{canFly}(x)]]] \tag{4.2}$$

$$(\forall k)[\text{TravelContext}(k) \supset \text{NormalConditions}(k)] \tag{4.3}$$

Example 2. Another example of projection contexts is that in which there is a parameter suppression. The class of contexts to which the *Above Theory* described below applies are static and, consequently, they do not contemplate the idea of situation at all. However, the contexts to which the theory of *Blocks* applies need to specify the situation at which a particular predicate holds and they do this by adding a situation parameter to the predicates *on* and *above*. Therefore, the meaning of these predicates is subject to two kinds of interpretations, namely the *AboveThInt* and the *BlocksInt*. While the former will assign to these symbols a type $d \times d$, the latter will assign to them a type $d \times d \times k$.

$$(\forall k)[\text{AboveTheory}(k) \supset k : [(\forall x, y)[\text{“on”}_{\text{AboveThInt}}(x, y) \supset \text{“above”}_{\text{AboveThInt}}(x, y)]]] \tag{4.4}$$

$$(\forall k)[\text{AboveTheory}(k) \supset k : [(\forall x, y)[\text{“above”}_{\text{AboveThInt}}(x, y) \wedge \text{“above”}_{\text{AboveThInt}}(x, y) \supset \text{“above”}_{\text{AboveThInt}}(x, y)]]] \tag{4.5}$$

$$(\forall k_1, k_2)[\text{Blocks}(k_1) \wedge \text{AboveTheory}(k_2) \supset (\forall x, y)[k_1 : [\text{“on”}_{\text{BlocksInt}}(x, y, k_2)] \equiv k_2 : [\text{“on”}_{\text{AboveThInt}}(x, y)]]] \tag{4.6}$$

$$(\forall k_1, k_2)[\text{Blocks}(k_1) \wedge \text{AboveTheory}(k_2) \supset (\forall x, y)[k_1 : [\text{“above”}_{\text{BlocksInt}}(x, y, k_2)] \equiv k_2 : [\text{“above”}_{\text{AboveThInt}}(x, y)]]] \tag{4.7}$$

Example 3. This last example is focused on the case of ambiguity contexts, and more precisely, on the problems with the homonymy of the predicate *Bank*. The context k_1 uses the predicate *Bank* but it does not specify under which interpretation, so that initially it could be understood both as a financial and a river bank. However, in the outermost context it is stated that if somebody withdraws money from a bank then the correct interpretation of the predicate *Bank* is that of a financial bank. Like in the previous example, the ability of our logic to quantify over interpretations facilitates the formalization of this example in an elegant manner.

$$k_1 : (\exists x, i)[\text{“}Bank\text{”}_i(x) \wedge At(Jon, x) \wedge gotMoney(Jon, x)] \tag{4.8}$$

$$(\forall k, x, y)[k : [gotMoney(x, y) \supset \text{“}Bank\text{”}_{FinancialInt}(y)]] \tag{4.9}$$

5 Related Work

In line with the work of Kaplan [15] and Montague [21], the Contextual Intensional Logic (CIL) [26] proposes a type-theoretic account of contexts, based on the introduction of a new type for contexts into the type theory of Intensional Logic [21]. Although the sorts of our system and the types of CIL are similar, they constitute very different approaches to the problem of the formalization of context.

In our view, CIL presents some limitations that are fundamental desiderata of a logic of context. First, it ignores the notion of local vocabulary and consequently the set of wffs is the same in every context. Second, the typification of

the symbols is rigid and not subject to interpretations. Third, it does not seem clear why the extension of a term under a particular index and context should be limited to a single individual. We think that contexts should be able to tolerate ambiguous denotations, what in our logic is achieved by differentiating between external and internal negations.

6 Conclusions

In this paper we have presented a model-theoretic semantics for the formalization of contexts based on a cognitive approach to the concepts of context and meaning. The logic has proved to be successful in addressing the quantificational case in a flexible manner that avoids the many counterintuitive problems and restrictions of its predecessors.

In our opinion, one of the main reasons why the previous initiatives are so limited is that they fail to differentiate between contexts and interpretations. We think that this differentiation is not only desirable from a technical perspective but it is also a conceptual requisite of a satisfactory formalization of contexts. In this sense, we have claimed that a cognitive approach to meaning as an embodied function is necessary to solve this problem. These intuitions have been materialized in what we have come to call a *logic of context and interpretation*, which not only handles contexts but also interpretations as first-class citizens. As has been shown in the examples, in contrast to the previous logics of contexts, our approach allows the formalization of the most challenging examples found in the literature.

Our future work will be focused on developing a sound and complete axiomatization of the system and studying the modeling of lexical ambiguity in our semantics. We also intend to research into how to accommodate temporal concepts, like events or actions, in our formalism.

References

1. Akman, V., Surav, M.: Steps Toward Formalizing Context. AI Magazine 17(3), 55–72 (1996)
2. Barwise, J., Perry, J.: Situations and Attitudes. MIT Press, Cambridge (1983)
3. Bouquet, P., Ghidini, C., Giunchiglia, F., Blanzieri, E.: Theories and Uses of Context in Knowledge Representation and Reasoning. Journal of Pragmatics 35(3), 455–484 (2003)
4. Buvac, S.: Quantificational Logic of Context. In: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996), pp. 600–606. AAAI Press, Menlo Park (1996)
5. Buvac, S., Mason, I.A.: Propositional Logic of Context. In: Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 1993), pp. 412–419. AAAI Press, Menlo Park (1993)
6. Devlin, K.: Chap. Situation Theory and Situation Semantics. In: Handbook of the History of Logic. Logic and the Modalities in the Twentieth Century, vol. 7, pp. 601–664. Elsevier, Amsterdam (2006)

7. Fauconnier, G.: Mental Spaces: Aspects of Meaning Construction in Natural Language. MIT Press, Cambridge (1985)
8. Gallese, V., Lakoff, G.: The Brain's Concepts: The Role of the Sensory-motor System in Conceptual Knowledge. Cognitive Neuropsychology 22(3-4), 455–479 (2005)
9. Ghidini, C., Giunchiglia, F.: Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence 127(2), 221–259 (2001)
10. Ghidini, C., Serafini, L.: Distributed First Order Logics. In: Frontiers of Combining Systems, vol. 2, pp. 121–139. Research Studies Press (2000)
11. Giunchiglia, F., Serafini, L.: Multilanguage Hierarchical Logics or: How we can do without Modal Logics. Artificial Intelligence 65(1), 29–70 (1994)
12. Guha, R., McCarthy, J.: Varieties of Context. In: Blackburn, P., Ghidini, C., Turner, R.M., Giunchiglia, F. (eds.) CONTEXT 2003. LNCS, vol. 2680, pp. 164–177. Springer, Heidelberg (2003)
13. Guha, R.V.: Contexts: a formalization and some applications. PhD thesis, Stanford University, California, USA (1992)
14. Hayes, P.J.: Contexts in Context. In: AAAI Fall Symposium of Context in Knowledge Representation and Natural Language. AAAI Press, Menlo Park (1997)
15. Kaplan, D.: On the Logic of Demonstratives. Journal of Philosophical Logic 8, 81–98 (1978)
16. Kosslyn, S.M.: Mental Images and the Brain. Cognitive Neuropsychology 22, 333–347 (2005)
17. Lewis, D.K.: Index, Context, and Content. In: Philosophy and Grammar, pp. 79–100. D. Reidel, Dordrecht (1980)
18. Lewis, D.K.: On the Plurality of Worlds. Blackwell Publishing, Malden (1986)
19. McCarthy, J.: Chap. Generality in Artificial Intelligence. In: Formalizing Common Sense: Papers by John McCarthy, pp. 226–236. Ablex Publishing, Greenwich (1990)
20. McCarthy, J.: Notes on Formalizing Contexts. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 1993), pp. 555–560. Morgan Kaufmann, San Francisco (1993)
21. Montague, R.: Formal Philosophy: Selected Papers of Richard Montague. Yale University Press, New Haven (1974)
22. Nardi, D., Brachman, R.J.: An Introduction to Description Logics. In: Description Logic Handbook, pp. 5–44. Cambridge University Press, Cambridge (2002)
23. Quine, W.V.O.: Propositional Objects. In: Ontological Relativity and Other Essays (J.Dewey Essays in Philosophy), pp. 139–160. Columbia University Press, New York (1969)
24. Serafini, L., Bouquet, P.: Comparing Formal Theories of Context in AI. Artificial Intelligence 155(1–2), 41–67 (2004)
25. Simons, P.: Parts: A Study in Ontology. Oxford University Press, Oxford (2003)
26. Thomason, R.H.: Type Theoretic Foundations for Context, Part 1: Contexts as Complex Type-Theoretic Objects. In: Bouquet, P., Serafini, L., Brézillon, P., Benercetti, M., Castellani, F. (eds.) CONTEXT 1999. LNCS (LNAI), vol. 1688, pp. 352–374. Springer, Heidelberg (1999)

Definition and Formalization of Entity Resolution Functions for Everyday Information Integration

David W. Archer and Lois M. L. Delcambre

Department of Computer Science
Portland State University
Portland, OR 97207
(darcher,lmd)@cs.pdx.edu

Abstract. Data integration on a human-manageable scale, by users without database expertise, is a more common activity than integration of large databases. Users often gather fine-grained data and organize it in an entity-centric way, developing tables of information regarding real-world objects, ideas, or people. Often, they do this by copying and pasting bits of data from e-mails, databases, or text files into a spreadsheet. During this process, users evolve their notions of entities and attributes. They combine sets of entities or attributes, split them again, update attribute values, and retract those updates. These functions are neither well supported by current tools, nor formally well understood. Our research seeks to capture and make explicit the data integration decisions made during these activities. In this paper, we formally define entity resolution and de-resolution, and show that these functions behave predictably and intuitively in the presence of attribute value updates.

1 Introduction

The term data integration evokes images of large-scale projects by experts in database administration. In previous work [1], we noted that a much more common form of data integration happens as part of everyday user tasks. End users often gather and integrate task-specific data from a variety of sources: web pages, text files, spreadsheets, e-mail messages, and relational databases. The data is often gathered into a tabular format such as a spreadsheet, where rows represent entities of interest and columns represent characteristics of or information about these entities.

The resulting table is often refined during integration as the user makes decisions. Entities may be merged (resolution) or split (de-resolution). New attributes may be added to the table. Attribute values may be updated. Each of these decisions refines the conceptual model imposed by the user on the data. We seek to capture the imposition of schema and the integration decisions expressed during everyday data integration. Our long-term goal is to exploit the knowledge expressed in these actions in order to help solve the general information integration problem. Along the way, we seek to

1. make the task of gathering, organizing, and using data from various sources easier,
2. track provenance, refinement, and deployment of gathered information to provide a rich semantic context to the user, and

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 126–142, 2008.

3. support re-use and sharing of expressed user integration actions and knowledge.

In this paper, we contribute a formalization of the resolution, de-resolution, and update functions, and show that they can be composed in useful ways. This definition allows us to maintain, traverse, and selectively reverse a history of resolution decisions and updates.

2 Formalizing the CHIME Model

We have previously reported [1] on a prototype end-user application called CHIME (Capturing Human Intension Metadata with Entities) that supports the goals outlined above. CHIME supports creation and manipulation of a single, entity-centric virtual table. Rows in this table correspond to entity instances, while columns correspond to attributes of the entities. In this paper, we use the term "entity" to refer to an entity instance. In addition to entity resolution/de-resolution, attribute resolution/de-resolution, and direct updates to attribute values, CHIME provides the following:

- Copy and paste capability that includes the selected sub-document, its provenance, related context found within the source document, and an address, or mark [14] with which to re-visit the selection in its source context
- On-the-fly schema definition and entity creation
- Automatic creation of groups of entities
- Export of the created relation in a variety of formats
- Injection of selected attributes or entire entity records into other work products
- Retention of integration decision history and reasoning

In order to have a sound theoretical basis for our work, we wish to show that the incremental refinement of an entity-centric relation, as done in CHIME, behaves predictably and intuitively. In the remainder of this paper, we formalize the CHIME model by outlining our assumptions, describing a conceptual model, defining a number of terms, proposing and proving a number of lemmas, and proving two key theorems.

We assume that each entity resolution operation is binary, that is, exactly two *parents* participate in the generation of the resolved *child*, contributing attribute values to the child as specified in a user-supplied *choice vector*. We claim without proof that n-way entity resolution may be achieved by iterative binary resolution. We also assume that each entity resolution operation is atomic. That is, if tuples t_1 and t_2 are the parents of a resolved entity represented in tuple t_3, then t_1 and t_2 cannot be the "parents" of any other resolved entity. We assume that each de-resolution operation is also atomic. That is, if we de-resolve t_3 into t_2 and t_1, t_3 leaves the relation and cannot de-resolve into some other tuple pair. Finally, we assume that users may make updates to table rows at any time, each update affecting exactly one attribute in one row. Any update of a group of rows and columns can be accomplished by a sequence of such updates.

Let R be a relation schema, with instance r, which represents entities and their characteristics, and consists of the following attributes:

- **KeyVal**, a monotonically increasing, integer-valued identifier that functions as a candidate key for R
- {**Attr1, Attr2,** …**AttrN**}, a set of one or more user-visible attributes with user-assigned names in R
- **Visible**, a Boolean attribute that specifies whether a given tuple in r is currently visible to the user and eligible for resolutions and updates. Visible is initially True for entities in r prior to any entity resolution
- {**Parent1, Parent2**}, a set of attributes in R that are foreign keys referencing KeyVal in R. These attributes specify the KeyVals of the parents of a tuple, if the tuple has been created by resolution, and are both null otherwise

Let H be a relation schema, with instance h, which represents the history of changes to attribute values due to resolutions and updates. H has the following attributes:

- **SequenceNum**, a monotonically increasing, integer-valued candidate key for H
- **KeyValue**, an integer-valued foreign key referencing R
- **AttrName**, an attribute with domain {Attr1, Attr2, … AttrN} (attributes in R)
- **AttrVal**, the value given attribute AttrName in the row in r with key KeyVal
- **ParentID**, a foreign key referencing R, though it may also take on the value {null}
- **Comment**, a string

As an example of resolution using R and H, suppose we define R to be {KeyVal, Name, EyeColor, ShoeSize, Visible, $Parent_1$, $Parent_2$} and construct r initially as

{1,	Bob,	blue,	12,	True,	null,	null}
{2,	Robert,	gray,	11,	True,	null,	null}
{3,	Sally,	brown,	6,	True,	null,	null}

Suppose we then decide that Bob and Robert are really the same person who prefers to be called Bob, but whose eyes are actually gray and who wears size 11 shoes. We would then update r to produce a relation r' with schema R as follows:

{1,	Bob,	blue,	12,	**False**,	null,	null}
{2,	Robert,	gray,	11,	**False**,	null,	null}
{3,	Sally,	brown,	6,	True,	null,	null}
{4,	**Bob,**	**gray,**	**11,**	**True,**	**1,**	**2**}

To represent the resolution decision, we record these changes in h as follows:

{1, 4,	Name,	Bob,	1,	"Bob = Robert, prefers Bob"}
{2, 4,	EyeColor,	gray,	2,	"Verified Bob's eyes are gray"}
{3, 4,	ShoeSize,	11,	2,	"Bob reports shoes are size 11, not 12"}

For a single resolution, these additions to h record the choice vector, indicating the parent from which each resolved attribute in the child is selected. In addition, h can be seen as a change log, and may be used in reverse sequence in an "undo" script.

We begin formalizing the CHIME model by defining a few key concepts. A *valid relation* r with schema R consists of a set of tuples, each of which represents an entity of interest. Tuples may be either *unresolved*, or *resolved*. An unresolved tuple is one

that has been added to r directly to represent an entity. A resolved tuple is one that has been created by merging two other tuples as described above. A valid relation has these properties:

- If r contains a tuple t_1 in which $Parent_1$ and Parent $_2$ are non-null (that is, t_1 is a resolved tuple), the tuples in r with KeyVal equal to the values of $Parent_1$ and $Parent_2$ in t_1 must have their Visible flag set to False: that is, both parents of a resolved tuple are always non-Visible.
- If r contains resolved tuples, then the attribute values for each user-visible attribute in all such tuples must be equal to the value of the same attribute in one or the other parent, unless some later update changes an attribute value.
- Both $Parent_1$ and $Parent_2$ must be either non-null and distinct, or null, in each tuple in r. That is, tuples have either no parents (i.e., are unresolved) or two distinct parents.
- The active domain of $Parent_1$ and $Parent_2$ does not include the KeyVal of the tuple in which they appear. That is, no tuple may be its own parent.

Let r be an instance of R such that two visible user-specified tuples, t_1 and t_2, in r are to be resolved into a single visible tuple in r' in an operation we call *entity resolution*. Suppose that k_1 and k_2 are the Keyval values for t_1 and t_2, respectively. We define t_3 as the result of a Cartesian product of tuples t_1 and t_2 with temporary schema R' = {$Keyval_1$ = k_1, $Keyval_2$= k_2, $Attr1_1$, $Attr1_2$, …$Attrn_1$, $Attrn_2$}, which is then reduced to schema R using the projection operator that returns one value from each parental pair in R' according to the user-supplied choice vector. The Parent attributes of t_3 are then filled in with the Keyvals of t_1 and t_2. The new instance r' contains:

- all tuples originally in r but not including t_1 and t_2
- two new tuples identical to t_1 and t_2, but with the Visible attribute set to False
- a new tuple t_3, which has a new key, Visible = True, $Parent_1$ and $Parent_2$ set to the keys of t_1 and t_2, respectively, and one value for each attribute Attr1…AttrN, selected by the user (as indicated in the choice vector) from the matching attributes in t1 and t2.

In relational algebra, we describe the entity resolution operation as shown in Figure 1. Values k_1 and k_2 above are user-specified. That is, the user selects the two tuples in r to be resolved. The underlined projection predicate "choices" in the final union term above represents the user's preferences for attribute values for the resolved entity. We use the *drop projection* operator from the extended relational algebra proposed by Wyss and Robertson [15], which we denote Ц. This operator has the semantics of specifying which attributes to eliminate during a projection, rather than which to retain.

We define the attributes of R as $R_{internal} \cup R_{external}$, where $R_{internal}$ = {KeyVal, Visible, $Parent_1$, $Parent_2$} and Rexternal = {$Attr_1$, $Attr_2$, … $Attr_n$}.
$R_{internal}$ is the set of attributes in R that represent internal state for record-keeping, while $R_{external}$ is the set of user-visible attributes that describe the entities in r. We denote the schema R_{cart}, that is, the schema of the Cartesian product ($\sigma_{Keyval=k1}$ r × $\sigma_{Keyval=k2}$ r) after removal of $R_{internal}$ as

$$R_{cart} = \{k_1.Attr_1, k_1.Attr_2, \ldots k_1.Attr_n, k_2.Attr_1, k_2.Attr_2, \ldots k_2.Attr_n\}.$$

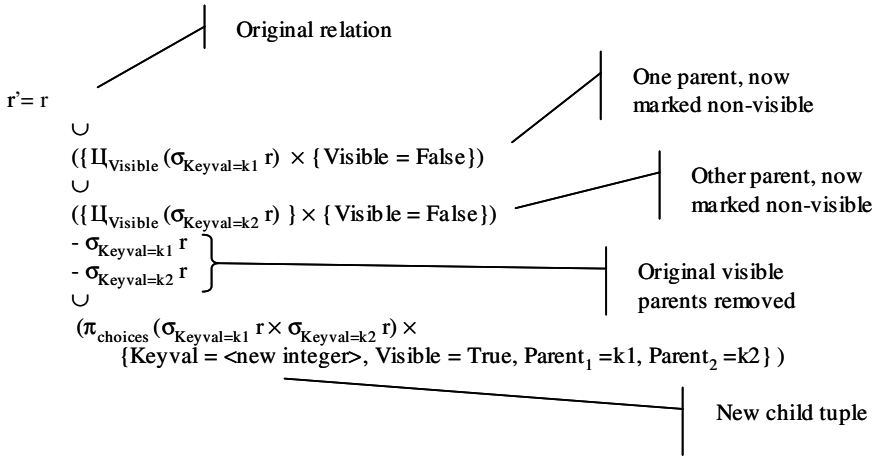


Fig. 1. Definition of Entity Resolution

R_{cart} is a schema that includes all of the external attributes from both entities being resolved. We define *choices* as a set of attribute names, such that choices = $\{x_i \mid x_i = k_1.Attr_i$ or $x_i = k_2.Attr_i$ for i = 1..n$\}$, that is, there must be exactly one attribute name specified in choices from each pair of attribute names corresponding to the same attribute in Parent1 and Parent2.

As we construct t_3 from the list of attributes available in t_1 and t_2, we also construct additions to h. Recall that h has schema {SequenceNum, Keyval, AttrName, AttrVal, ParentID, Comment}. For each attribute a_i in choices, we add a new tuple to h, with

- a monotonically increasing value for SequenceNum
- the candidate key of the newly created tuple in r'
- the name of the attribute being specified
- the value assigned to that attribute in the new tuple
- the Keyval of the parent tuple from which this value was taken
- a comment string where the user describes the reason for this choice

In relational algebra, we describe changes to h as shown in Figure 2.

We denote the entity resolution of two tuples t_1 and t_2 in r, resulting in t_3 in r' as $P_{1,2}$. By this definition, any legal entity resolution results in a valid relation.

Entity de-resolution is the process of reversing an earlier entity resolution decision. This implies removing the selected, resolved entity from an instance r', and re-instantiating the two parent tuples of the entity by making them both visible again.

Let r' be an instance of R such that two user-specified tuples t_1 and t_2 in a prior instance r were resolved into a single tuple t_3 in r'. Suppose that k_3 is the value of Keyval for t_3. Note that by the definition of entity resolution given above, the tuple with candidate key k_3 has the $Parent_1$ attribute set to t_1 and $Parent_2$ set to t_2. Also, note that if the Parent attributes are null, it indicates that the selected tuple has not been

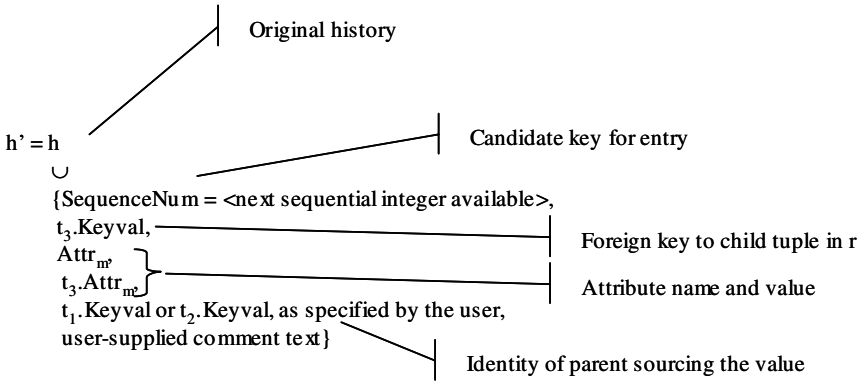


Fig. 2. Additions to h due to entity resolution in r. One tuple is added to h for each user-visible attribute a_i in R, i=1…n.

resolved from others in the relation, so no de-resolution is possible. When we de-resolve a selected resolved tuple in r', we create a new instance r, which contains all tuples originally in r' unrelated to the de-resolution of t_3, and two new tuples identical to t_1 and t_2 (the non-visible parent tuples of t_3), but with the Visible attribute set to True. The new instance r has had removed from it the tuple t_3 and the two non-visible tuples t_1 and t_2.

In relational algebra, we describe entity de-resolution effects on r as shown in Figure 3.

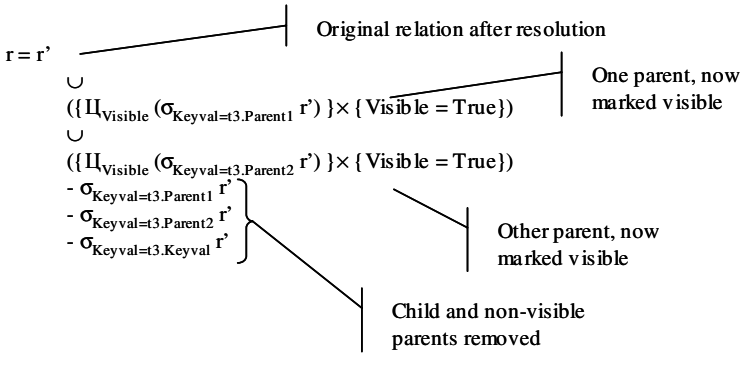


Fig. 3. Definition of Entity De-resolution

This new relation r contains exactly the same tuples as the original (pre-resolution) r, so de-resolution is the functional inverse of entity resolution as defined above.

As we de-resolve t_3 into t_1 and t_2, we also modify h by removing all tuples added to h when the resolution $P_{1,2}$ was performed. In relational algebra,

$$h = h' - \sigma_{Keyval=t3}\ h'$$

We choose here to remove such tuples, retaining only "good" decisions in our history table. We could equally well choose to retain the history of "poor" decisions, and the history of un-doing them, perhaps in another relation, or in the history table h, flagged with a new attribute. We denote de-resolution of a child t_3 into t_1 and t_2, and associated changes to h, as Δ_3.

An *update* is a change to the value of an external attribute in a user-specified tuple in r. Updates may be made to either unresolved or resolved tuples, so long as they are visible. The latter we term post-resolution updates. Functionally, both kinds of updates are equivalent. As an example, suppose we have relations r' and h' from our previous example:

```
r' = {{1,  Bob,     blue,    12,     False,  null,  null},
      {2,  Robert,  gray,    11,     False,  null,  null},
      {3,  Sally,   brown,   6,      True,   null,  null},
      {4,  Bob,     gray,    11,     True,   1,     2}}

h' = {{1,  4, Name,       Bob,   1,   "Bob = Robert, prefers Bob"},
      {2,  4, EyeColor,   gray,  2,   "Verified Bob's eyes are gray"},
      {3,  4, ShoeSize,   11,    2,   "Bob reports shoes are size
11"}}
```

Now suppose that the user finds that Bob's shoe size is actually 9 rather than 11, and makes the appropriate update. That is, the user changes Bob's shoe size from the originally resolved value of 11 to the value 9. Then we have a new table

```
r'' = {{1,    Bob,     blue,    12,     False,  null,  null},
       {2,    Robert,  gray,    11,     False,  null,  null},
       {3,    Sally,   brown,   6,      True,   null,  null},
       {4,    Bob,     gray,    9,      True,   1,     2}}
```

and we record this update in our history table as

```
h'' = {{1, 4,     Name,       Bob,   1,  "Bob = Robert, prefers Bob"},
       {2, 4,     EyeColor,   gray,  2,  "Verified Bob's eyes are gray"},
       {3, 4,     ShoeSize,   11,    2,   "Bob reports shoes are size 11"},
       {4, 4,     ShoeSize,   9,     null, "Fixed Bob's shoe size!!!"}
```

In relational algebra, we define updates to the relation r' as:

$$r'' = r' \cup (\Pi_{AttrM} (\sigma_{Keyval=K} r') \times \{AttrM = NewValue\}) - \sigma_{Keyval=K} r'$$

where the Keyval K, the attribute name "AttrM", and the value "NewVal" are supplied by the user. Subsequent updates may revise or undo the effect of earlier updates. A later update that undoes a prior update, we call a retraction. We denote an update to tuple t_1 in r, along with the associated changes to h, as $\Theta_{1,\ attrname,\ value}$.

De-resolution of a resolved entity might be interpreted in several ways. The simple approach, discussed above, is to assume that all decisions made with respect to the resolved entity, including both the original resolution decision and all subsequent

updates, should simply be discarded. However, this approach ignores any updates made after resolution. For example, consider once again the relation r'':

r'' = {{1, Bob, blue, 12, False, null, null},
{2, Robert, gray, 11, False, null, null},
{3, Sally, brown, 6, True, null, null},
{4, Bob, gray, 9, True, 1, 2}}

Suppose that after changing Bob's shoe size to 9 from 11, as shown in the last tuple of h'', we decide that Robert and Bob are really not the same person after all. Because of this, we wish to remove the tuple with KeyVal 4 from r'' and restore its parents. However, suppose the change in shoe size was correct: 11 really should be 9 for one of the parents. We would like to avoid requiring tedious modification of the original relation r to represent this correctly. Upon de-resolution, we should back-propagate the value 9 into the attribute that currently has the value 11, because the shoe size of Robert (11) was preferred when we resolved Bob and Robert. After back-propagation of updates, we would like the final state of the relation to be

r = {{1, Bob, blue, 12, True, null, null},
{2, Robert, gray, **9**, True, null, null},
{3, Sally, brown, 6, True, null, null}}

To perform de-resolution in the presence of later updates to a resolved tuple, we first perform simple de-resolution. Next, we modify the resulting relation by application of the post-resolution updates to the appropriate parent attributes. We implement de-resolution in relational algebra as follows. Suppose we wish to de-resolve a resolved tuple t_3 with candidate key k_3. First, we join history table rows representing the choice vector for t_3 with the rows representing any post-resolution updates for attributes in t_3.

$$h_1 = \sigma_{\text{Keyval=k3, Parent} \neq \text{null}} h \overset{\text{KeyVal, attrName}}{\not\bowtie} \sigma_{\text{Parent =null}} h$$

The important attributes in each result row in h_1 are

- the name of the attribute updated (attrName)
- the order in which each update was done (right.SequenceNum)
- the value assigned by the update (right.attrVal)
- the key of the parent that should receive the back-propagated value (left.Parent)

Next, we reduce this result to retain only these important attributes:

$$h_2 = \pi_{\text{right.SequenceNum, attrName, left.Parent, right.attrVal}}(h_1)$$

At this point, we have a set of all attributes that have been changed in t3, the identities of the tuples into which these changes should be back-propagated[1], the values to back-propagate, and the order in which these updates should be applied. However, only the last update for each attribute in the selected tuple matters, because all others were overwritten by their successors. This suggests the next step: group by the attribute

[1] For clarity of presentation, we assume here that the correct target tuple for back-propagation of an update is the parent tuple from which the original attribute value in the child was taken. An equally valid choice would be to choose the other parent. In general, this selection should be made with user input.

name that was changed, and find the most recent update in each group. We apply the group-by operator γ and the aggregation function max() to achieve this step. We simplify the expression by defining max to return the tuple containing the maximum value of the specified attribute rather than simply the value itself:

$$h_3 = \gamma_{attrName,\ max(Sequencenum)}\ h_2$$

Next, we can discard the ordering information:

$$h_4 = \pi_{attrName,\ Parent,\ attrVal}\ h_3$$

h4 consists of at most a single tuple per attribute in $R_{external}$ because we selected changes for only a single resolved tuple when forming h_1, and in h_3 we filtered those down to only the most recent updates. We transform the resulting set of updates by moving the value of attrName attributes to become the names of the attrVal attributes via the pivot operator proposed by Wyss and Robertson [15]:

$$h_5 = PIVOT_{attrVal \rightarrow attrName}\ h_4$$

h5 contains at most two tuples, one for each parent, with values for each of the attributes to be back-propagated into that parent, and null for all other attribute values in h4.

Next, we join r with h5 on r.KeyVal = h5.Parent. We define this join, denoted]✎, to preserve all attributes from the original parent tuple in r, except that we replace those with matching non-null attributes in h5 with the values from h5. For this purpose, we define a variant of equi-join that has the following semantics. In addition to the usual join behavior, the join retains attributes with their corresponding attribute values from *only the left* relation whenever the attribute is null in the right relation, and retains attributes with their corresponding attribute values from *only the right* relation whenever the right relation has a non-null value. We then project away the name of the parent, to yield the new version of the parent tuple with schema identical to R. We denote this step

$$r_{temp} = \text{Ц}_{h5.Parent}\ (r\ ]\text{✎}_{KeyVal=Parent}\ h_5)$$

Finally, we union this with r and then remove the old parent tuples

$$r' = (r \cup r_{temp}) - r\ \text{✂}_{keyval=parent}\ h_5$$

We now describe the changes to h required for this operation. As with simple de-resolution, all entries in h resulting from the resolution operation must be removed. We must also remove all entries in h that describe post-resolution updates to the resolved tuple. We describe the removal of all these updates from h algebraically as

$$h = h' - \sigma_{Keyval=t3}\ h'$$

We denote de-resolution of a child t_3 with key k3, with back-propagation of later updates into its parents t_1 and t_2, along with associated changes to h, as Δ^{b}_{k3}. The remainder of this paper considers de-resolution as defined this way.

A *legal sequence* of resolutions, updates, and de-resolutions has two requirements. First, all tuples in r referred to as targets of updates and/or sources of resolution and de-resolution have not been made non-visible by a prior operation in the sequence. Second, all changes to r result in a valid relation at each step (including the end of the sequence).

Two entity resolutions are *independent* if the tuples they operate on are disjoint. That is, neither operation relies on the other for creation of a tuple involved in its operation, and the two operations do not specify the same tuple as a parent. Two update operations are independent if they refer to either different tuples, or different attributes, or both. A resolution and an update are independent if the parents and child of the resolution are disjoint from the target tuple of the update.

3 Lemmas and Theorems

As users integrate data for everyday tasks, their understanding of the data set evolves. Decisions made at one point often must be reversed later as a user's understanding improves, as in the example of Robert and Bob in Section 2. Unfortunately, users are likely to learn of their errors a long time (during which they made many other decisions) after committing them. It is undesirable for a tool to insist that users undo all intervening decisions, remove the erroneous decision, and then redo everything. We prefer to define an integration tool that provides convenient correction of such errors. In order to deliver a useful tool, we must first know whether such convenience can be concomitant with correct results. In this section, we show that a user may undo an earlier resolution or update regardless of intervening operations, providing that the operations form a legal sequence as defined above. Figure 4 summarizes the reasoning presented in this section.

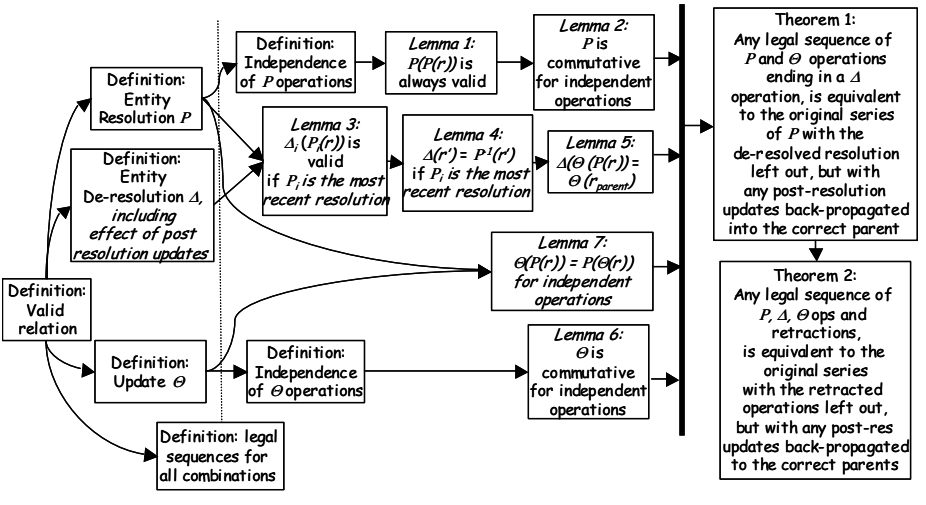


Fig. 4. Reasoning Diagram for Definitions, Propositions, and Theorems

Lemma 1: Composition of resolutions in a legal sequence yields a valid relation

Suppose we have two entity resolutions $P_{1,2}$ yielding t_3 and $P_{4,5}$ yielding t_6 for distinct t_1, t_2, t_4, and t_5 in r. Then either the two resolutions are independent, or they are not. If they are independent, then applying $P_{1,2}$ will result in a valid r' by definition, and subsequently applying $P_{4,5}$ to the valid relation r', such that the two operations form a

legal sequence, will also result in a valid relation, because t_4 and t_5 are distinct from t_1 and t_2, the only tuples made ineligible for further resolution in r'. If the two resolutions $P_{1,2}$ and $P_{4,5}$ are not independent, then we have two possibilities: either one of t_1 and t_2 must be the same as t_4 or t_5, which violates our definition of resolution because both t_1 and t_2 are already parents in r'; or t_3 must be the same as t_4 or t_5. In the latter case, composition yields a valid relation per our definition.

Lemma 2: Composition of two independent resolution operations is commutative

Suppose r = $\{t_1, t_2, t_4, t_5\}$ and all of t_1, t_2, t_4, and t_5 have Visible = true. That is, they all meet the requirements for use as source tuples in entity resolution operations. Also suppose that $P_{1,2}$ yielding t_3 and $P_{4,5}$ yielding t_6 are independent resolution operations. From our definition of entity resolution, we know that $P_{1,2}$ results in the addition of t_3 to r, and the change of t_1 and t_2 to be non-visible, such that they cannot participate in future resolution. Similarly, we know that $P_{4,5}$ results in the addition of t_6 to r, and the change of t_4 and t_5 to be non-visible. If we perform $P_{1,2}$ first and $P_{4,5}$ second, we have a valid relation (from Lemma 1) r' = $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ such that t_1, t_2, t_4, and t_5 are non-visible and $\{t_3, t_6\}$ are visible. If we perform these resolutions in the opposite order, we have a valid relation r'' = $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ such that t_1, t_2, t_4, and t_5 are non-visible and $\{t_3, t_6\}$ are visible. Because r' = r'', the two operations are commutative.

Lemma 3: Entity de-resolution of a child results in a valid relation if the most recent resolution on a relation is the one being de-resolved, and there are no intervening updates.

Suppose we have a relation r' which resulted from the resolution of two tuples in a valid relation r, where t1 is the tuple created by the resolution resulting in r', and Parent1 and Parent2 are the tuples that were resolved to create t1. Then immediately after the resolution, r' is valid, and contains the new tuple t1 marked visible, and the tuples Parent1 and Parent2, marked not visible. Now suppose we de-resolve t1, resulting in relation r''. Between the resolution and de-resolution, we assume that no other resolutions occur. This means that the only possible changes to r' between the pair of operations is addition of new tuples which by definition have both parent fields set to null. The de-resolution of t1, resulting in r'', eliminates t1 and restores Parent1 and Parent2 to Visible, but affects no other tuples and does not affect any fields except the Visible field in Parent1 and Parent2. Then we see that upon de-resolution, r must be valid, because the only resolved tuples in r'' are those that existed in the original r, which by definition meet the first and second stipulations of a valid relation since r was valid.

Lemma 4: Entity de-resolution of a child is the inverse operation of the entity resolution of its parents, provided that it was the most recent resolution and that there were no intervening updates

Our definition of entity resolution tells us that

$$
\begin{aligned}
r' = r \\
&\cup (\{\Pi_{Visible} (\sigma_{Keyval=k1} r) \} \times \{Visible = False\}) \\
&\cup (\{\Pi_{Visible} (\sigma_{Keyval=k2} r) \} \times \{Visible = False\}) \\
&- \sigma_{Keyval=k1} r \\
&- \sigma_{Keyval=k2} r
\end{aligned}
$$

$$\cup\ (\pi_{\underline{choices}}\ (\sigma_{Keyval=k1}\ r \times \sigma_{Keyval=k2}\ r)$$
$$\times\ \{Keyval = <integer>,\ Visible = True,\ Parent1 = k1,\ Parent2 = k2\}\ )$$

Substituting for r using our definition of de-resolution (shown in italics below), we have

$$r' = \textit{(r'}$$
$$\textit{}\cup(\{\amalg_{Visible}\ (\sigma_{Keyval=t3.Parent1}\ r')\}\ \times \{Visible = True\})$$
$$\textit{}\cup(\{\amalg_{Visible}\ (\sigma_{Keyval=t3.Parent2}\ r')\}\ \times \{Visible = True\})$$
$$\textit{}- \sigma_{Keyval=t3.Parent1}\ r'$$
$$\textit{}- \sigma_{Keyval=t3.Parent2}\ r'$$
$$\textit{}- \sigma_{Keyval=t3.Keyval}\ r'\ )$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=k1}\ r)\}\ \times\ \{Visible = False\})$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=k2}\ r)\}\ \times\ \{Visible = False\})$$
$$-\ \sigma_{Keyval=k1}\ r$$
$$-\ \sigma_{Keyval=k2}\ r$$
$$\cup\ (\pi_{\underline{choices}}\ (\sigma_{Keyval=k1}\ r \times \sigma_{Keyval=k2}\ r)$$
$$\times\ \{Keyval = <new\ integer>,\ Visible = True,$$
$$Parent1 = \pi_{Keyval}\ (\sigma_{Keyval=k1}\ r),\ Parent2 = \pi_{Keyval}(\sigma_{Keyval=k2}\ r)\})$$

Note that $\sigma_{Keyval=t3.Keyval}\ r'$ shown above is simply the newly resolved tuple we added to r, because t3.Keyval is a candidate key, and because the tuple with Keyval of t3.Keyval is guaranteed to be present as a result of the original entity resolution. The tuple with t3.Keyval as key is:

$$(\pi_{\underline{choices}}\ (\sigma_{Keyval=k1}\ r \times \sigma_{Keyval=k2}\ r)$$
$$\times$$
$$\{Keyval = <integer>,\ Visible = True,\ Parent1 = \pi_{Keyval}\ (\sigma_{Keyval=k1}\ r),$$
$$Parent2 = \pi_{Keyval}(\sigma_{Keyval=k2}\ r)\}\ )$$

This allows us to cancel[2] these two terms, leaving us with

$$(r'$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=t3.Parent1}\ r')\}\ \times \{Visible = True\})$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=t3.Parent2}\ r')\}\ \times \{Visible = True\})$$
$$-\ \sigma_{Keyval=t3.Parent1}\ r'$$
$$-\ \sigma_{Keyval=t3.Parent2}\ r')$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=k1}\ r)\}\ \times\ \{Visible = False\})$$
$$\cup\ (\{\amalg_{Visible}\ (\sigma_{Keyval=k2}\ r)\}\ \times \{Visible = False\})$$
$$-\ \sigma_{Keyval=k1}\ r$$
$$-\ \sigma_{Keyval=k2}\ r$$

Next, note that

$$(\{\amalg_{Visible}\ (\sigma_{Keyval=t3.Parent1}\ r')\}\ \times\ \{Visible = True\})$$

is a tuple in r' with the key of one parent of t3, and visible set to True. Also note that

[2] We assume here that the Keyval shown here (as being generated at the time the resolution was performed) matches the Keyval of the tuple chosen for deresolution.

$$(\{\amalg_{\text{Visible}} (\sigma_{\text{Keyval=t3.Parent2}} r')\} \times \{\text{Visible} = \text{True}\})$$

is a tuple in r' with candidate key of the other parent of t3, also with the visible flag set to True. This pair is identical to the tuple pair $\{\sigma_{\text{Keyval=k1}} r, \sigma_{\text{Keyval=k2}} r\}$, the original parent tuples of t3. As a result, we can cancel these 4 terms, leaving us with

$$\begin{aligned}
&(r' \\
&\quad - \sigma_{\text{Keyval=t3.Parent1}} r' \\
&\quad - \sigma_{\text{Keyval=t3.Parent2}} r') \\
&\cup (\{\amalg_{\text{Visible}} (\sigma_{\text{Keyval=k1}} r)\} \times \{\text{Visible} = \text{False}\}) \\
&\cup (\{\amalg_{\text{Visible}} (\sigma_{\text{Keyval=k2}} r)\} \times \{\text{Visible} = \text{False}\})
\end{aligned}$$

Now note that $\sigma_{\text{Keyval=t3.Parent1}} r'$ results in a single tuple, one of the parents of t3, with the Visible flag set to False as a result of the original resolution step, and $\sigma_{\text{Keyval=t3.Parent2}} r'$ is the other parent. These are the same as the pair

$$\begin{aligned}
&(\{\amalg_{\text{Visible}} (\sigma_{\text{Keyval=k1}} r)\} \times \{\text{Visible} = \text{False}\}) \\
&\cup (\{\amalg_{\text{Visible}} (\sigma_{\text{Keyval=k2}} r)\} \times \{\text{Visible} = \text{False}\})
\end{aligned}$$

allowing us to cancel these terms. This leaves us with simply r' = r', proving that de-resolution is the inverse of resolution.

Lemma 5: De-resolution with update back-propagation is equivalent to modifying the parents in the original relation

Lemma 4 shows that entity resolution followed immediately by a matching de-resolution effectively restores the relation to its state prior to the resolution. We now show that we can achieve the desired behavior of back-propagation during de-resolution by showing that intervening updates are pushed back into the appropriate parents.

Recall from our definition that the *choices* vector from a resolution is used during the corresponding de-resolution to specify which parent is populated with values from post-resolution updates. That is, during de-resolution, each post-resolution update is performed on one parent or the other, depending on the choices vector. The set of tuple selections expressed by the user in *choices* is the same set that the user would express if making updates directly to the parent tuples if no resolution had been done. As a result, back-propagation updates the same attributes in the same tuples to the same values that would be specified had the user made direct updates, using the same choices, without the intervening resolution.

Lemma 6: Composition of two independent update operations is commutative

Two independent updates must either target different tuples in r, or different attributes in R, or both. Because of this, two independent updates can always be applied in either order, resulting in the same relation. As a result, the two operations are commutative.

Lemma 7: Composition of a resolution and an independent update in a legal sequence is commutative

If an entity resolution and an update are independent and form a legal sequence, then either the resolution follows the update, but does not involve the updated tuple, or the

update follows the resolution, but does not target any of the tuples involved in the resolution. In either case, the result is identical, so the operations are commutative.

Theorem 1: Any legal sequence of entity resolutions and updates ending in a de-resolution is equivalent to the original series of resolutions and updates with the de-resolved resolution left out, but with any post-resolution updates to the de-resolved entity back-propagated, i.e., applied to the correct parent tuple according to the choices vector used during the original resolution

If the sequence is legal, then the terminal de-resolution operation must apply to some resolution operation that is independent of any other intervening resolutions between the two. Because we know that independent resolutions commute (Lemma 2), and that independent updates and resolutions commute (Lemma 7), and that independent updates commute (Lemma 6), we commute these operations until the resolution of interest is only separated from its de-resolution by a sequence of dependent updates. In effect, we "push" the affected resolution down towards the end of the sequence, pushing ahead of it any dependent updates, until the end of the sequence consists of the resolution, any updates to the resolved tuple, and the de-resolution. Because updates affect only user-visible attributes, and because entity de-resolution depends only on internal attributes, intervening updates between the resolution – de-resolution pair do not prevent correct de-resolution, though they will affect which values are back-propagated to parents during de-resolution. We then apply Lemma 5, obtaining a relation that has all post-resolution updates applied (back-propagated) to the correct parent, which is equivalent to applying the desired updates to the parent in the original relation.

Theorem 2: Any legal sequence of entity resolutions, updates, de-resolutions, and retractions of updates is equivalent to the original sequence of resolutions and updates with the de-resolved and retracted operations left out, but with post-resolution updates to the de-resolved entities applied to the correct parents, according to the choices vector used during resolution. In practical terms, the final result of a set of operations and corrections is the same as the original set without the errors, but with updates back-propagated appropriately.

Suppose we have a list of resolutions, updates, and de-resolutions that we wish to apply to relation r, with associated effects on relation h. We begin by applying the first of these, generating a new relation pair r' and h'. We continue to apply these, one at a time, until we encounter a de-resolution. Then we apply Theorem 1 to return the sequence to a sequence of resolutions and updates without the undone resolution, but with all updates applied to the right parents. Then we proceed by applying subsequent operations to this valid relation from our legal sequence, until encountering another de-resolution. Theorem 1 can then be applied again, and so on. We repeat this incremental application of Theorem 1 as often as required, each time obtaining a correct intermediate result, eventually yielding a correct final result.

Retractions of updates in the sequence are simply updates that happen to supercede a previous update. Because of this, the above reasoning extends to cover legal sequences of resolutions, de-resolutions, updates, and retractions.

4 Related Work

Research that we know of in entity resolution (also called record linkage [7] and de-duplication [8]) is largely focused on identifying entity matches, rather than on how merges are done and decision histories retained and traversed. As early as 1969 [3], work was published on statistical methods for determining entity matches. More recently, machine learning and other automated approaches have been explored [9, 10, 11, 12]. Garcia-Molina [13] provides a comprehensive overview of this area. Other recent approaches involve gathering semantic information from users or user dataspaces. MOBS [4] seeks to retrieve information about entities from diverse sources, given initial seed data. It depends on explicit contributions to its mapping database by a community of users. Our long-term vision is similar to MOBS, but our approach emphasizes gathering metadata behind the scenes as users gather data for their own purposes. In addition, our work emphasizes post-integration tasks like entity resolution, which is beyond the scope of MOBS. SEMEX [5] uses a pre-defined but extensible ontology to construct an entity-centric, logical view of a user's desktop by constructing a database of objects and associations between them. In this way, SEMEX provides access to data stored in multiple applications without imposing a data organization that is application-centric. The SEMEX approach differs from ours in that SEMEX takes in a user's entire personal information space to identify entities and associated data, while we focus on user selections and actions to derive schema and entity evolution.

Garcia-Molina and the Swoosh team [6] distinguish two entity resolution functions: *match* and *merge*. Match is a black box representing the disambiguation methods discussed above. The merge function has to do with modifying a relation to take into account an identified match. The authors provide a conceptual data model, describe the merge function formally, and prove several properties of compositions of merges. Our entity resolution function differs from Swoosh in that Swoosh assumes a simple domination model, while our model accounts for a true merge of characteristics from both entities. Also, Swoosh retains no parentage or history: a dominated entity is immediately removed from the relation and no history is kept. Our model retains both parent entities of the merge, as well as a history of which attribute values were chosen for expression in the child. Swoosh does not address de-resolution. Our model fully supports reversal of resolution decisions. Finally, Swoosh considers only resolution, while we consider interactions between resolutions, de-resolutions, and updates.

5 Conclusions and Future Work

We observed previously [1] that everyday data integration by users who are not data management experts is ubiquitous. In this paper, we have identified a set of integration decisions and associated user actions common to these tasks. We presented a formalism, and used it to show that the key post-integration actions of entity resolution, updates, retractions, and de-resolutions behave predictably and intuitively.

We have implemented an emulator for CHIME and a data integration application based on the CHIME model. We wrote the emulator in Haskell, a polymorphically

typed, pure functional language. Haskell's polymorphism makes abstractions easy, so that we could focus on the integration functions rather than the implementation of the emulator. The ease of recursion in functional languages also made data manipulation in the emulator easy. We used the emulator to develop a model of entity and attribute resolution and de-resolution, and to explore composition of these functions. We wrote our prototype application, also called CHIME, in J# as a Windows application. With CHIME and support from the SPARCE middleware package [2], users can gather data from a variety of source documents: Excel spreadsheets, Word documents, web browsers, and other applications.

We have more formal work to do. For example, attribute resolution and de-resolution must be added to our framework. Once our formal development is complete, we will pursue a more complete application suite. Then we can explore the next step: exploiting the captured end-user integration information to aid large-scale information integration.

References

[1] Archer, D., Delcambre, L.: Capturing Users' Everyday, Implicit Information Integration Decisions. In: Conferences in Research and Practice in Information Technology, Auckland, NZ, vol. 83, pp. 133–138 (2007)

[2] Murthy, S., Maier, D., Delcambre, L., Bowers, S.: Putting Integrated Information in Context: Superimposing Conceptual Models with SPARCE. In: Proceedings of the First Asia-Pacific Conference on Conceptual Modeling, Dunedin, NZ, pp. 71–80 (2004)

[3] Fellegi, I., Sunter, A.: A theory for record linkage. Journal of the Americal Statistical Association 64, 1183–1210 (1969)

[4] Sayyadian, M., Shakery, A., Doan, A., Zhai, C.: Toward Entity Retrieval over Structured and Text Data. In: Proceedings of the first Workshop on the Integration of Information Retrieval and Databases, Sheffield, UK (2004)

[5] Cai, Y., Dong, X., Halevy, A., Liu, J., Madhavan, J.: Personal Information Management with SEMEX. In: Proceedings of the ACM SIGMOD International Conference on Management of Data Baltimore (2005)

[6] Benjelloun, O., Garcia-Molina, H., Su, Q., Widom, J.: Swoosh: a generic approach to entity resolution. Technical Report 2005-5. Stanford University, Palo Alto (2005)

[7] Winkler, W.: Matching and record linkage. In: Cox, B. (ed.) Business Survey Methods. Wiley, New York (1995)

[8] Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vancouver, Canada (2002)

[9] Neiling, M., Jurk, S.: The object identification framework. In: Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington, DC, pp. 33–40 (2003)

[10] Mann, G., Yarowsky, D.: Unsupervised personal name disambiguation. In: Proceedings of the Conference on Computational Natural Language Learning, Edmonton, Canada, pp. 33–40 (2003)

[11] Hsuing, P., Moore, A., Neill, D., Schneider, J.: Alias detection in link data sets. In: Proceedings of the International Conference on Intelligence Analysis, McLean, VA (2005)

[12] Malin, B.: Unsupervised name disambiguation via social network similarity. In: Proceedings of the SIAM Workshop on Link Analysis, Counterterrorism, and Security, Newport Beach, CA, pp. 93–102 (2005)
[13] Garcia-Molina, H.: Entity resolution: Overview and challenges. In: Proceedings of the International Conference on Conceptual Modeling, Shanghai, China, pp. 1–2 (2004)
[14] Delcambre, L., Maier, D., Bowers, S., Weaver, M., Deng, L., Gorman, P., Ash, J., Lavelle, M., Lyman, J.: Bundles in Captivity: An Application of Superimposed Information. In: Proceedings of the 17th International Conference on Data Engineering, pp. 111–120 (2001)
[15] Wyss, C., Robertson, E.: A Formal Characterization of PIVOT/UNPIVOT. In: Proceedings of CIKM 2005, Bremen, Germany, pp. 602–608 (2005)

A Theoretical Framework for the Declarative Debugging of Datalog Programs

R. Caballero[1], Y. García-Ruiz[1], and F. Sáenz-Pérez[2,⋆]

[1] Departamento de Sistemas Informáticos y Computación,
[2] Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
rafa@sip.ucm.es, ygarciar@fdi.ucm.es, fernan@sip.ucm.es

Abstract. The logic programming language Datalog has been extensively researched as a query language for deductive databases. Although similar to Prolog, the Datalog operational mechanisms are more intricate, leading to computations quite hard to debug by traditional approaches. In this paper, we present a theoretical framework for debugging Datalog programs based on the ideas of declarative debugging. In our setting, a debugging session starts when the user detects an unexpected answer for some query, and ends with the debugger pointing to either an erroneous predicate or to a set of mutually recursive predicates as the cause of the unexpected answer. Instead of representing the computations by means of trees, as usual in declarative debugging, we propose graphs as a more convenient structure in the case of Datalog, proving formally the soundness and completeness of the debugging technique. We also present a debugging tool implemented in the publicly available deductive database system DES following this theoretical framework.

1 Introduction

Deductive databases rely on logic programming based query languages. Although not very well-known out of the academic institutions, some of their concepts are used in today relational databases to support advanced features of more recent SQL standards, and even implemented in major systems (e.g., the linear recursion provided in IBM's DB2 following the SQL-99 standard). A successful language for deductive databases has been Datalog [1], which allows users writing more expressive queries than relational databases. Relations and queries in Datalog are considered from a model-theoretic point of view, that is, thinking of relations as sets, and the language itself as a tool for manipulating sets and obtaining answer sets.

Raising the abstraction level generally implies a more complex computation mechanism acting as a black-box hidden from the user. Although this leads to more expressive programs, it also makes query debugging a very difficult

⋆ The authors have been partially supported by the Spanish National Project MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS-CAM(S-0505/TIC/0407).

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 143–159, 2008.

process. An operational semantics oriented debugger is not helpful in this context, since the underlying computational mechanism is not directly related to the model-theoretic approach, but to implementation techniques such as *magic sets* [2] or *tabling* [3]. The few existing proposals for debugging specifically Datalog programs are usually based on "imperative" debugging, that try to follow the computation model to find bugs. These proposals are mainly based on forests of proof trees [4,5,6], which makes debugging a trace based task not so amenable to users. The first work related to the declarative debugging of Datalog programs is due to [7], but a variant of SLD resolution is used in order to look for program errors, imposing to traverse at least as many trees as particular answers are obtained for any query.

In the more general setting of *answer set programming* [8], there have been several proposals for diagnosing program errors in the last few years. In [9] a technique for detecting *conflict sets* is proposed. The paper explains how this approach can be used for detecting missing answers. Our proposal is limited to a more particular type of programs, namely stratified programs, but it can be applied for diagnosing not only missing but also *wrong* answers. In [10] the authors propose a technique that transforms programs into other programs with answer sets including debugging-relevant information about the original programs. This approach can be seen as a different, complementary view of the debugging technique described here.

In [11] we proposed a novel way of applying declarative debugging (also called *algorithmic debugging*, a term first coined in the logic programming field by E.H. Shapiro [12]), to Datalog programs. In that work, we introduced the notion of *computation graphs* (shortly *CG*s) as a suitable structure for representing and debugging Datalog computations. One of the virtues of declarative debugging is that it allows theoretical reasoning about the adequacy of the proposal. This paper addresses this task, proving formally the soundness and completeness of the debugging technique. We also present a prototype based in these ideas and included as part of a publicly available Datalog system DES [13].

The next section introduces the theoretical background needed for proving the properties of the debugger. Section 3 presents the concept of computation graph and proves several properties of *CG*s, while Section 4 includes the soundness and completeness results. Section 5 is devoted to discuss some implementation issues. Finally, Section 6 summarizes the work and presents the conclusions.

2 Datalog Programs

In this section, we introduce the syntax and semantics of Datalog programs and define the different types of errors that can occur in our setting. Although there are different proposals for this language, we will restrict our presentation to the language features included in the system DES [13]. Observe that the setting for Datalog presented here is a subsumed by the more general framework of *Answer Set Programming* [8].

2.1 Datalog Syntax

We consider (recursive) Datalog programs [14,15], i.e., normal logic programs without function symbols. In our setting, *terms* are either variables or constant symbols and *atoms* are of the form $p(t_1, \ldots, t_n)$, with p an n-ary predicate symbol and t_i terms for each $1 \leq i \leq n$. The notation $t_1, \ldots, t_n$ will be usually abbreviated as $\bar{t}_n$. A *positive* literal is an atom, and a *negative* literal is a negated atom. A negated atom is syntactically constructed as $not(A)$, where A is an atom. The atom contained in a literal L will be denoted as *atom(L)*. The set of variables of any formula F will be denoted as $var(F)$. A formula F is *ground* when $var(F) = \emptyset$.

A *rule* (or *clause* in the logic programming context) R has the form $p(\bar{t}_n)$:− $l_1, \ldots, l_m$ representing the first order logic formula $p(\bar{t}_n) \leftarrow l_1 \wedge \ldots \wedge l_m$, where l_i are literals for $i = 1 \ldots m$, and $m \geq 0$. The left-hand side atom $p(\bar{t}_n)$ will be referred to as the *head* of R, the right-hand side $l_1, \ldots, l_n$ as the *body* of R, and the literals l_i as *subqueries*. The variables occurring only in the body $l_1 \wedge \ldots \wedge l_m$ are assumed to be existentially quantified and the rest universally quantified. We require that $vars(H) \subseteq vars(B)$ for every program rule $H :- B$. A *fact* is a rule with empty body and ground head. The symbol :− is dropped in this case. The *definition of a relation* (or *predicate*) p in a program P consists of all the program rules with p in the head. A *query* (or *goal*) is a literal.

We consider stratified negation, a form of negation introduced in the context of deductive databases in [16]. A program P is called *stratified* if there is a partition $\{P_1, \ldots, P_n\}$ of P s.t. for $i = 1 \ldots n$:

1. If a relation symbol occurs in a positive literal of the body of any rule in P_i then its definition is contained in $\cup_{j \leq i} P_j$.
2. If a relation symbol occurs in a negative literal of the body of any rule in P_i then its definition is contained in $\cup_{j < i} P_j$.

We call each P_i a *stratum*. For instance, consider the Datalog program of Figure 1. We can check that the program is stratified by defining two strata: P_1 containing the rules for `star`, `orbits` and `intermediate`, and P_2 containing the rule for `planet`.

```
star(sun).
orbits(earth, sun).
orbits(moon, earth).
orbits(X,Y)       :- orbits(X,Z), orbits(Z,Y).
planet(X)         :- orbits(X,Y), star(Y), not(intermediate(X,Y)).
intermediate(X,Y) :- orbits(X,Y), orbits(Z,Y).
```

Fig. 1. A (buggy) Datalog Program

2.2 Program Models

We consider *Herbrand interpretations* and *Herbrand models*, i.e., Herbrand interpretations that make every Herbrand instance of the program rules logically true formulae. An *instance* of a formula is the result of applying the substitution θ to a formula F. We use the notation $F\theta$ instead of $\theta(F)$ for representing instances. The set *Subst* represents the set of all the possible substitutions. Often, we will be interested in *ground instances of a rule*, assuming implicitly that every rule is renamed with new variables each time it is selected. The composition operation between substitutions is defined in the usual way and fulfilling the property $(F\sigma)\theta = F(\sigma \cdot \theta)$ for all $\sigma, \theta \in Subst$. Two formulae φ, φ' are *variants* if $\varphi = \varphi'\theta$, where θ is a renaming, i.e., a bijection among variables. We say that $\sigma \in Subst$ is an *instance* of $\theta \in Subst$ when $\sigma = \theta\mu$, with μ some substitution. In this case, we write $\sigma \geq \theta$.

Given a Herbrand interpretation I for a the Datalog program P, we use the notation $I \models F$ to indicate that the formula F is true in I. The *meaning of a query* Q w.r.t. the interpretation I, denoted by Q_I, is the set of ground instances $Q\theta$ s.t. $I \models Q\theta$. If Q is an atom, then an equivalent definition is $Q_I = \{Q\theta \mid Q\theta \in I \text{ for some } \theta \in Subst\}$.

In logic programming without negation, the existence of a *least Herbrand model* for every program P is ensured, and it can be obtained as the least fixed point of a closure operator T_P, which is defined over any interpretation I as:

$$A \in T_P(I) \text{ iff for some rule } (H :- B) \in P, I \models B\theta \text{ and } A = H\theta$$

In these conditions, the least Herbrand model is defined as $T_P \uparrow \omega(\emptyset)$, i.e., as the fixed point obtained when iterating the operator starting at the empty interpretation. In general, however, the existence of the least Herbrand model is not ensured in programs using negation. Fortunately, due to the use of stratified programs in Datalog, the existence of a so-called *standard model*, which we will represent also as $\mathcal{M}$, is in any case ensured [14]. Given a program P stratified by the partition $\{P_1, \ldots, P_k\}$, we define the sets $M_0 = \emptyset$, $M_1 = T_{P_1} \uparrow \omega(M_0)$, $\ldots$, $M_k = T_{P_k} \uparrow \omega(M_k - 1)$. Then, the standard model of P is defined as $\mathcal{M} = M_k$. The standard model verifies the following properties (the proofs can be found in [14]):

Proposition 1. *Let P be a program stratified by the partition $\{P_1, \ldots, P_k\}$. Then:*

1. *$\mathcal{M}$ is a minimal model.*
2. *$\mathcal{M}$ is supported, i.e., for all $p(\bar{s}_n) \in \mathcal{M}$ there exists an* associated program rule *$(H :- B) \in P$ and an* associated substitution *$\theta \in Subst$ such that $p(\bar{s}_n) = H\theta$, $\mathcal{M} \models B\theta$ and $B\theta$ ground (due to our safety condition, $var(H) \subseteq var(B)$, which means that $H\theta$ is also ground).*
3. *Conversely, if there is some $(H :- B) \in P$, $\theta \in Subst$ s.t. $\mathcal{M} \models B\theta$, then $\mathcal{M} \models H\theta$.*
4. *$\mathcal{M}$ is independent of the stratification.*

5. *The following chain of inclusions holds:*

$$\begin{array}{llll} \emptyset = M_0 & \subseteq T_{P_1}(M_0) & \subseteq T^2_{P_1}(M_0) & \subseteq \ldots \subseteq M_1 \\ M_1 & \subseteq T_{P_2}(M_1) & \subseteq T^2_{P_2}(M_2) & \subseteq \ldots \subseteq M_2 \\ & & \ldots & \\ M_{k-1} & \subseteq T_{P_k}(M_{k-1}) & \subseteq T^2_{P_k}(M_{k-1}) & \subseteq \ldots \subseteq M_k = \mathcal{M} \end{array}$$

Since functions are not allowed in Datalog, the standard model is finite and it can be actually computed. In fact, the deductive database systems such as DES are implemented to obtain the values $Q_{\mathcal{M}}$ for every query Q. Thus, $Q_{\mathcal{M}}$ will be referred to as the *answer* to Q. From now on, we assume that the Datalog system supporting the debugger verifies this condition, which is a reasonable requirement in the context of Datalog. This is different from the general setting of logic languages such as Prolog, even if we restrict to the case of Prolog programs without functions in the signature. For instance, consider the following dummy program:

```
p(X) :- q(X).      q(X) :- p(X).
```

The program is valid both in Prolog and in Datalog. However, the goal (resp. query) `p(X)` shows the difference between the two settings: In Prolog, it leads to a non-terminating computation, whereas in Datalog it succeeds with the answer {}, meaning that no ground instance of `p(X)` can be deduced from the program. Our selected system DES computes the answer to a query following a top-down approach, so that only the relevant information to obtain $Q_{\mathcal{M}}$ is computed in order to increase the efficiency of the computation.

The concept of standard model above is generalized by that of *stable model* [17], which can be applied also to non-stratified programs. However, in this work we restrict our semantics to stratified programs because this is a requirement of several Datalog systems.

2.3 Correct and Incorrect Programs

We use the term *intended interpretation*, denoted by $\mathcal{I}$, to denote the Herbrand model the user has in mind for the program. If $\mathcal{M} = \mathcal{I}$, we say that the program is *well-defined*, and if $\mathcal{M} \neq \mathcal{I}$ we say that the program is *buggy*. Declarative debugging assumes that the user focus on query answers for comparing the intended interpretation to the standard Herbrand model actually computed. Thus, we say that $Q_{\mathcal{M}}$ is an *unexpected answer* for a query Q if $Q_{\mathcal{M}} \neq Q_{\mathcal{I}}$. An unexpected answer can be either a *wrong answer*, when there is some $Q\theta \in Q_{\mathcal{M}}$ s.t. $Q\theta \notin Q_{\mathcal{I}}$, or a *missing answer*, when there is $Q\theta \in Q_{\mathcal{I}}$ s.t. $Q\theta \notin Q_{\mathcal{M}}$. In the first case, $Q\theta$ is a *wrong instance*, while in the second one $Q\theta$ is a *missing instance*. Observe that an unexpected answer can be both missing and wrong at the same time. The next proposition indicates that an unexpected answer to a positive query implies an unexpected answer to its negation.

Proposition 2. *Let P be a program containing at least one constant, $\mathcal{I}$ its intended model and Q a positive query. Then, $Q_{\mathcal{M}}$ is a missing answer for Q iff*

$(\neg Q)_{\mathcal{M}}$ is a wrong answer for $\neg Q$, and $Q_{\mathcal{M}}$ is a wrong answer for Q iff $(\neg Q)_{\mathcal{M}}$ is a missing answer for $\neg Q$.

Proof. Straightforward from the definition of meaning of a query w.r.t. an interpretation, since $Q_I \cap (\neg Q)_I = \emptyset$ in every interpretation I. Then, $p(\bar{t}_n) \notin Q_{\mathcal{M}}$ and $p(\bar{t}_n) \in Q_{\mathcal{I}}$, i.e., if $p(\bar{t}_n)$ is a missing instance and $Q_{\mathcal{M}}$ is a missing answer, iff $p(\bar{t}_n) \in (\neg Q)_{\mathcal{M}}$), $p(\bar{t}_n) \notin (\neg Q)_{\mathcal{I}}$, i.e., $p(\bar{t}_n)$ is a wrong instance and $(\neg Q)_{\mathcal{M}}$ is a wrong answer for $\neg Q$. Analogous for the other case. □

An unexpected answer indicates that the program is erroneous, and it will be considered as the initial symptom for a user to start the debugging process. The two usual causes of errors considered in the declarative debugging of logic programs are *wrong* and *incomplete* relations:

Definition 1 (Wrong Relation). *Let P be a Datalog program. We say that $p \in P$ is a **wrong relation** w.r.t. $\mathcal{I}$ if there exist a rule variant $p(\bar{t}_n) :- l_1, \ldots, l_m$ in P and a substitution θ such that $\mathcal{I} \models l_i\theta$, $i = 1 \ldots m$ and $\mathcal{I} \not\models p(\bar{t}_n)\theta$.*

Definition 2 (Incomplete Relation). *Let P be a Datalog program. We say that $p \in P$ is an **incomplete relation** w.r.t. $\mathcal{I}$ if there exists an atom $p(\bar{s}_n)\theta$ s.t. $\mathcal{I} \models p(\bar{s}_n)\theta$ and, for each rule variant $p(\bar{t}_n) :- l_1, \ldots, l_m$ and substitution θ', either $p(\bar{t}_n)\theta' \neq p(\bar{s}_n)\theta$ or $\mathcal{I} \not\models l_i\theta'$ for some l_i, $1 \leq i \leq m$.*

In Datalog we also need to consider another possible cause of errors, namely the *incomplete set of relations*. This concept depends on the auxiliary definition of uncovered set of atoms.

Definition 3 (Uncovered Set of Atoms). *Let P be a Datalog program and $\mathcal{I}$ an intended interpretation for P. Let U be a set of atoms s.t. $\mathcal{I} \models p(\bar{s}_n)$ for each $p(\bar{s}_n) \in U$. We say that U is an **uncovered set of atoms** if for every rule $p(\bar{t}_n) :- l_1, \ldots, l_m$ in P and substitution θ s.t.:*

- *$p(\bar{t}_n)\theta \in U$,*
- *$\mathcal{I} \models l_i\theta$ for $i = 1 \ldots m$*

there is some $l_j\theta \in U$, $1 \leq j \leq m$, with l_j a positive literal.

Now, we are ready for defining the third kind of error, which generalizes the idea of incomplete relation:

Definition 4 (Incomplete Set of Relations). *Let P be a Datalog program and S a set of relations defined in P. We say that S is an **incomplete set of relations** in P iff exists an uncovered set of atoms U s.t. for each relation $p \in S$, $p(\bar{t}_n) \in U$ for some $t_1, \ldots, t_n$.*

To the best of our knowledge, this error has not been considered in the literature about Datalog debugging so far, but it is necessary for correctly diagnosing Datalog programs. Consider again the program `p(X):- q(X). q(X):-p(X).` with the intended interpretation $I = \{p(a), q(a)\}$ and the query `p(X)`. The computed

answer {} is a missing answer with `p(a)` as missing instance. However, neither of the two relations is incomplete, because their rules can produce the values `p(a)`, `q(a)` by means of the instance given by the substitution $\theta = \{X \mapsto a\}$. So, $U = \{p(a), q(a)\}$ is an uncovered set of atoms and hence $S = \{p, q\}$ is an incomplete set of relations.

We say that a relation is *buggy* when it is wrong, incomplete or member of an incomplete set of relations, and that it is well-defined otherwise. Observe that, due to the use of negation, a wrong answer does not correspond always to a wrong relation. For instance, in the following program:

```
p(X) :-  r(X), not(q(X)).
% missing q(a).
r(a).
```

with intended interpretation $\mathcal{I} = \{q(a), r(a)\}$ the query `p(X)` produces the wrong answer {`p(a)`} but there is no wrong relation in the program and instead there is an incomplete relation (q).

As an example, consider the program of Figure 1. This program defines a relation `orbits` by two facts and a rule establishing the transitive closure of the relation. A relation `star` is defined by one fact and indicates that the sun is a star. The relation `intermediate` is defined in terms of `orbits`, relating two bodies `X` and `Y` whenever there is some intermediate body between them. Finally, `planet` is defined as a body `X` that orbits directly a star `Y`, without any other body in between. However, a mistake has been introduced in the program: The underlined `Y` in the rule for `intermediate` should be `Z`. As a consequence, the query `planet(X)` yields the missing answer {} (assuming that the atom `planet(earth)` is in $\mathcal{I}$). In the next section, we will show how such errors can be detected by using declarative debugging based on computation graphs.

3 Computation Graphs

In this section, we define a structure for representing Datalog computations and prove their adequacy for declarative debugging.

3.1 Graph Terminology

We consider finite *directed graphs* $G = (V, E)$, where V is a finite set of vertices and E a finite set of directed edges, $E \subseteq V \times V$. Often, we use the notation $v \in G$ instead of $v \in V$ and $(u, v) \in G$ instead of $(u, v) \in E$. Given any vertex $u \in G$ we say that $v \in G$ is a *successor* of u in G if $(u, v) \in G$, which we represent by the notation $succ_G(u, v)$.

Given $G = (V, E)$, we say that $G' = (V', E')$ is a *subgraph* of G if G' is a graph s.t. $V' \subseteq V$ and $E' \subseteq E$. A particular case of subgraph is the *subgraph generated* from a subset of vertices $V' \subseteq V$. This subgraph is of the form $G' = (V', E')$, where $E' = \{(u, v) \in G \mid u, v \in V'\}$.

In a directed graph, the *output degree* of a vertex $v \in G$ is the cardinal of the set $\{u \in G \mid (v,u) \in G\}$ and it is represented by $gr^+_G(v)$. Analogously, the value $gr^-_G(v) = \mid \{u \in G \mid (u,v) \in G\} \mid$ represents the *input degree* of v. These concepts can be naturally extended to subgraphs by defining $gr^+_G(G') = \mid \{(u,v) \in G \mid u \in G', v \notin G'\} \mid$, $gr^-_G(G') = \mid \{(v,u) \in G \mid u \in G', v \notin G'\} \mid$. We remove the subindex G in gr^+_G, gr^-_G whenever the reference to the graph considered cannot be ambiguous in the context.

A sequence of vertices $u_1, u_2, \ldots, u_n$ of G such that $(u_i, u_{i+1}) \in G$ for all $i = 1 \ldots n-1$ are called a *walk* from u_1 to u_n. A walk s.t. $u_1 = u_n$ is called a *circuit*. A walk with no repeated vertices except maybe the first and the last vertex is called a *path*. If indeed $u_1 = u_n$ the path is called a *cycle*, i.e., a cycle is a special case of circuit with exactly one vertex repeated. The notation $path_G(u,v)$ represents a path starting at u and ending in v in some graph G.

A directed graph G is called *strongly connected* if, for every pair of vertices $u, v \in G$, there is a path from u to v and a path from v to u. The strongly connected components of a directed graph are its maximal strongly connected subgraphs, and they form a partition of G.

3.2 Datalog Computation Graphs

The *computation graph* (*CG* in short) for a query Q w.r.t. a program P is a directed graph $G = (V, E)$ such that each vertex V is of the form $(Q', Q'_{\mathcal{M}})$, where Q' is a subquery produced during the computation, and $Q'_{\mathcal{M}}$ is the computed answer for Q'. The next definition includes the construction of a computation graph. Observe that the answers of the subqueries are not relevant for the graph structure and, therefore, they are included as part of the vertices in a last step.

Definition 5 *(***Computation Graph***).* *Let P be a Datalog program and Q a query either of the form $p(\bar{a}_n)$ or $not(p(\bar{a}_n))$. The computation graph for Q w.r.t. P is represented by a pair (V, E) of vertices and edges defined as follows:*

The construction of the graph uses an auxiliary set A for containing the vertices that must be expanded in order to complete the graph.

1. *Put $V = A = \{p(\bar{a}_n)\}$ and $E = \emptyset$.*

2. *While $A \neq \emptyset$ do:*
   (a) *Select a vertex u in A with query $q(\bar{b}_n)$. $A = A \setminus \{u\}$.*
   (b) *For each rule R defining q, $R = (q(\bar{t}_n) :- l_1, \ldots, l_m)$ with $m > 0$, such that there exists $\theta = mgu(\bar{t}_n, \bar{b}_n)$, the debugger creates a set S of new vertices. Initially, we define $S = \emptyset$ and include new vertices associated to each literal l_i, $i = 1 \ldots m$ as follows:*
      i. *$i = 1$, a new vertex is included: $S = S \cup \{atom(l_1)\theta\}$.*
      ii. *$i > 1$. We consider the literal l_i. For each set of substitutions $\{\sigma_1, \ldots, \sigma_i\}$ with $dom(\sigma_1 \cdot \ldots \cdot \sigma_{i-1}) \subseteq var(l_1) \cup \cdots \cup var(l_i)$ such that for every $1 < j \leq i$:*
         – *$atom(l_{j-1})(\sigma_1 \cdot \ldots \cdot \sigma_{j-1}) \in S$, and*

– $l_{j-1}(\sigma_1 \cdot \ldots \cdot \sigma_j) \in (l_{j-1}(\sigma_1 \cdot \ldots \cdot \sigma_{j-1}))_{\mathcal{M}}$

include a new vertex in S*:*

$$S = S \cup \{atom(l_i)(\sigma_1 \cdot \ldots \cdot \sigma_i)\}$$

(c) For each vertex $v \in S$*, test whether there exists already a vertex* $v' \in V$ *such that* v *and* v' *are variants (i.e., there is a variable renaming). There are two possibilities:*

– *There is such a vertex* v'*. Then,* $E = E \cup \{(u, v')\}$*. That is, if the vertex already exists, we simply add a new edge from the selected vertex* u *to* v'*.*

– *Otherwise,* $V = V \cup \{v\}$*,* $A = A \cup \{v\}$*, and* $E = E \cup \{(u, v)\}$*.*

3. *Complete the vertices including the computed answer* $Q_{\mathcal{M}}$ *of every subquery* Q*.*

End of Definition

We will use the notation $[Q = Q_{\mathcal{M}_A}]$ for representing the content of the vertices. The values $Q_{\mathcal{M}_A}$ included at step 3 can be obtained from the underlying deductive database system by submitting each Q. The vertex is *valid* if $Q_{\mathcal{M}_A}$ is the expected answer for Q, and *invalid* otherwise.

Figure 2 shows the CG for the query `planet(X)` w.r.t. the program of Figure 1. The first vertex included in the graph at step 1 corresponds to `planet(X)`.

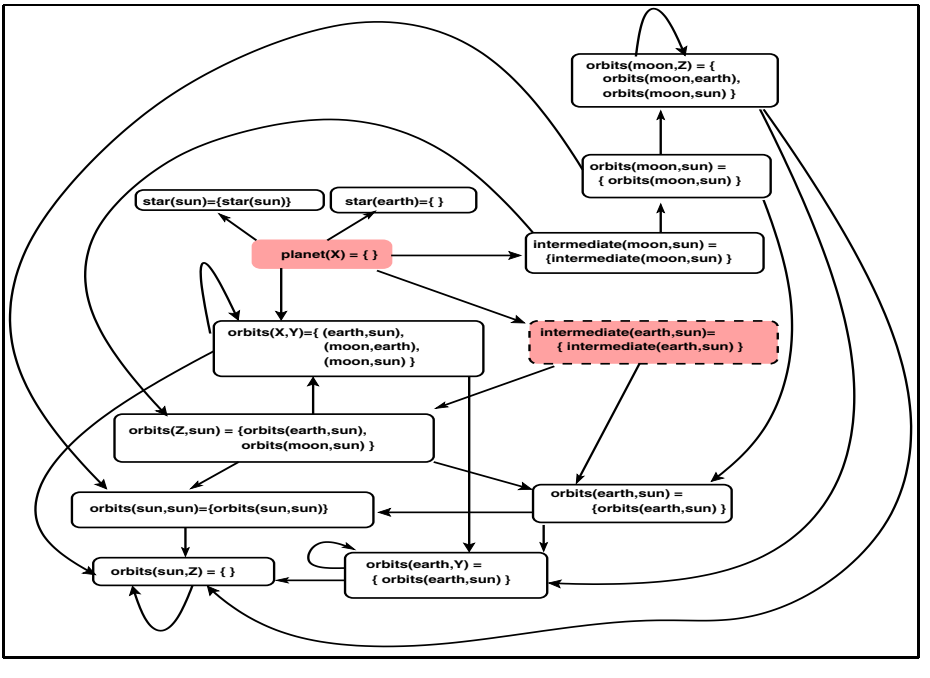


Fig. 2. CG for the Query `planet(X)` w.r.t. the Program of Figure 1

From this vertex and by using the only program rule for `planet`, four new vertices are added, the first one corresponding to the first literal `orbits(X,Y)`. Since two values of `Y` satisfy this subquery, namely `Y=sun` and `Y=earth`, the definition introduces two new vertices for the next literal `star(Y)`, `star(sun)` and `star(earth)`. The last one produces the empty answer, but `star(sun)` succeeds. Then, the last literal in the rule, `not(intermediate(X,Y))`, yields vertices for the two values of `X` and the only value of `Y` that satisfies the two previous literals. Observe, however, that the vertices for this literal are introduced in the graph without the negation, i.e., the CG will contain only subqueries for atoms. This simplifies the questions asked to the user during the navigation phase, and can be done without affecting the correctness of the technique because the validity of the positive literal implies the validity of its negation, and the other way round (although the type of associated error changes, see Proposition 2). The rest of the vertices of the example graph are built expanding the successors of `planet(X)` and repeating the process until no more vertices can be added.

The termination of the process is guaranteed because in our setting the signature is finite and the CG cannot have two occurrences of the same vertex due to step 2c, which introduces edges between existing vertices instead of creating new ones when possible.

The next proposition relates the elements of the computed answer stored at a vertex u with the immediate successors of u and vice versa.

Proposition 3. *Let $u = [Q = Q_{\mathcal{M}}]$ be a vertex in the computation graph G of some query w.r.t. a program P. Let $p(\bar{s}_n)$ be an instance of Q. Then $p(\bar{s}_n) \in Q_{\mathcal{M}}$ iff there exist a rule variant $p(\bar{t}_n) \ :- \ l_1, \ldots, l_m$ and a substitution θ such that among the successors of u in G there are vertices of the form $[atom(l_i)\sigma_i = A_i]$ with $\theta \geq \sigma_i$ for each $i = 1 \ldots m$.*

Proof. First, we suppose that $p(\bar{s}_n) \in Q_{\mathcal{M}}$. Let $(p(\bar{t}_n) \ :- \ l_1, \ldots, l_m) \in P$ and $\theta \in Subst$ be respectively the associated rule and the associated substitution to $p(\bar{s}_n)$, as defined in Proposition 1, item 2. Then, by this proposition, $p(\bar{s}_n) = p(\bar{t}_n)\theta$, which implies the existence of the $mgu(p(\bar{t}_n), p(\bar{s}_n))$ because we always consider rule variants and, hence, $var(p(\bar{s}_n)) \cap var(p(\bar{t}_n)) = \emptyset$. Then, the algorithm of Definition 5, item 2b, ensures that this program rule produces new vertices, successors of u in G. We check by induction on the number of literals in the body rule, m, that these vertices are of the form $[atom(l_i)\sigma_i = A_i]$ with $\theta \geq \sigma_i$ for $i = 1 \ldots m$. If $m = 0$, the result holds trivially. If $m = 1$, then there is a successor of u of the form $[atom(l_1)\theta = A_1]$ (item 2(b)i of Definition 5). For the inductive case $m > 1$, we assume that there is already a successor of u of the form $[atom(l_{m-1})\sigma = A_{m-1}]$, $\theta \geq \sigma$, i.e., $\theta = \sigma \cdot \sigma_m$ for some substitution σ_m. By the graph construction algorithm, σ must be of the form $\sigma = \sigma_1 \cdot \ldots \cdot \sigma_{m-1}$. By Proposition 1, item 2, $M \models l_{m-1}\theta$, i.e., $l_{m-1}\theta \in A_{m-1}$ (by the same Proposition 1, $l_{m-1}\theta$ is ground, and therefore must be part of the computed answer for $l_{m-1}\sigma$). Hence, $l_{m-1}(\sigma_1 \cdot \ldots \cdot \sigma_m) \in A_{m-1}$. In these conditions, the algorithm of Definition 5 includes a new successor of u with the form $atom(l_m)(\sigma_1 \cdot \ldots \cdot \sigma_m)$.

Conversely, if there exists a program rule, a substitution, and successor vertices as the proposition indicates, then it can be proved by a similar reasoning

that $M \models (l_1, \ldots, l_m)\theta$, and then, Proposition 1, item 3, ensures that $p(\bar{s}_n) = p(\bar{t}_n)\theta$ verifies $\mathcal{M} \models p(\bar{s}_n)$. In particular, if $p(\bar{s}_n)$ is ground, this means that $p(\bar{s}_n) \in \mathcal{M}$. □

The relation among a vertex and its descendants also relates the validity of them, as the following proposition states:

Proposition 4. *Let G be a computation graph and $u = [p(\bar{s}_n) = A]$ be an invalid vertex of G such that p is a well-defined relation. Then, u has some invalid successor v in G.*

Proof. If the vertex u is invalid, then A is either a wrong or a missing answer for $p(\bar{s}_n)$, which means that it contains either a wrong or a missing instance.

Suppose that $p(\bar{s}_n)\theta$ is a wrong instance for some $\theta \in subst$. Since $p(\bar{s}_n)\theta \in (p(\bar{s}_n))_{\mathcal{M}}$, by Proposition 1, there exists some associated program rule $R \in P$ and substitution θ' s.t. $(R)\theta' = (p(\bar{t}_n) \ :- \ l_1, \ldots l_m)\theta'$, with $\mathcal{M} \models l_i\theta'$ for all $i = 1 \ldots m$ and $p(\bar{t}_n)\theta' = p(\bar{s}_n)\theta$. From Proposition 3, it can be deduced that there are successor vertices of u of the form $[atom(l_i)\sigma_i = A_i]$ for all $i = 1 \ldots m$, with $\theta' \geq \sigma_i$. Assume that all these vertices are valid. Then, for each $i = 1 \ldots m$ we can ensure the validity of $l_i\theta'$ because:

- If l_i is a positive literal, from the validity of the answer for $atom(l_i)\sigma_i$ we obtain the validity of the more particular $atom(l_i)\theta'$ (the validity of a formula entails the validity of its instances).
- If l_i is a negative literal, from the validity of the answer for $atom(l_i)\sigma_i$ we obtain the validity of the answer for $atom(l_i)\theta'$, and from this, the validity of the answer for $l_i\theta'$ (as a consequence of Proposition 2).

Then, we have that $\mathcal{M} \models (l_1, \ldots, l_m)\theta'$, but $\mathcal{M} \not\models p(\bar{t}_n)\theta'$, i.e., $(R)\theta'$ is a wrong instance. But this is not possible because p is well-defined. Therefore, some of the successors of u must be invalid.

The proof is analogous in the case of a missing answer. □

3.3 Buggy Vertices and Buggy Circuits

In the traditional declarative debugging scheme [18] based on trees, program errors correspond to *buggy nodes*. In our setting, we also need the concept of buggy node, here called *buggy vertex*, but in addition our computation graphs can include *buggy circuits*:

Definition 6 (Buggy Circuit). *Let $CG = (V, A)$ be a computation graph. We define a* buggy circuit *as a circuit $W = v_1 \ldots v_n$ s.t. for all $1 \leq i \leq n$:*

1. *v_i is invalid.*
2. *If $(v_i, u) \in A$ and u is invalid then $u \in W$.*

Definition 7 (Buggy Vertex). *A vertex is called buggy when it is invalid but all its successors are valid.*

The next result proves that a computation graph corresponding to an initial error symptom, i.e., including some invalid vertex, contains either a buggy circuit or a buggy vertex.

Proposition 5. *Let G be a computation graph containing an invalid vertex. Then, G contains either a buggy vertex or a buggy circuit.*

Proof. Let G be the computation graph and $u \in G$ an invalid vertex. From G, we obtain a new graph G' by including all the invalid vertices reachable from u. More formally, G' is the subgraph of G generated by the set of vertices

$$\{v \in G \mid \text{ there is a path } \Pi = path_G(u, v) \text{ and } w \text{ invalid for every } w \in \Pi\}$$

Now, we consider the set S of strongly connected components in G',

$$S = \{C \mid C \text{ is a strongly connected component of } G'\}$$

The cardinality of S is finite since G' is finite. Then, there must exist $C \in S$ such that $gr^+_{G'}(C) = 0$. Moreover, for all $u \in C$, $succ_G(u, u')$ means that either $u' \in C$ or u' is valid because $u' \notin C$, u' invalid, would imply $gr^+_{G'}(C) > 0$. Observe also that, by the construction of G', every $u \in C$ is invalid. Then:

- If C contains a single vertex u, then u is a buggy vertex in G.
- If C contains more than a vertex, then all its vertices form a buggy circuit in G. □

4 Soundness and Completeness

The debugging process we propose can be summarized as follows:

1. The user finds out an unexpected answer for some query Q w.r.t. some program P.
2. The debugger builds the computation graph G for Q w.r.t. P.
3. The graph is traversed, asking questions to the user about the validity of some vertices until a buggy vertex or a buggy circuit has been found.
4. If a buggy vertex is found, its associated relation is pointed out as buggy. If instead a buggy circuit is found, the set of relations involved in the circuit are shown to the user indicating that at least one of them is buggy or that the set is incomplete.

Now, we must check that the technique is reliable, i.e., that it is both sound and complete. First we need some auxiliary lemmata.

Lemma 1. *Let G be a computation graph for some query Q w.r.t. a program P, and let $C = u_1, \ldots, u_k$, with $u_k = u_1$ be a circuit in G. Then, all the u_i are of the form $[Q_i = Q_{i\mathcal{M}}]$ with Q_i associated to a positive literal in its corresponding program rule for $i = 1 \ldots k - 1$.*

Proof. It can be proved that every relation occurring in some Q_i depends recursively on itself. This means that Q_i cannot occur negatively in a clause because this would mean than P is not stratified (see Lemma 1 in [14]). □

Lemma 2. *Let $v = [p(\bar{s}_n) = \dots]$ be a vertex of some CG G obtained w.r.t. some program P with standard model $\mathcal{M}$. Let $p(\bar{t}_n) :- l_1, \dots, l_m$ be a rule in P, and θ s.t. $p(\bar{t}_n)\theta = p(\bar{s}_n)\theta$, and that $\mathcal{M} \models l_1\theta, \dots, l_k\theta$ for some $1 \leq k \leq m$. Then, v has children vertices in G of the form $[atom(l_i)\theta_i = \dots]$ for $i = 1 \dots k+1$, with $\theta \geq \theta_i$.*

Proof. The proof corresponds to that of Proposition 3, but considering only the first $k+1$ literals of the program rule. □

Observe that theoretically the debugger could be applied to any computation graph even if there is no initial wrong or missing answer. The following soundness result ensures that in any case it will behave correctly.

Proposition 6 (Soundness). *Let P be a Datalog program, Q be a query and G be the computation graph for Q w.r.t. P. Then:*

1. *Every buggy node in G is associated to a buggy relation.*
2. *Every buggy circuit in G contains either a vertex with an associated buggy relation or an incomplete set of relations.*

Proof

1. Suppose that G contains a buggy vertex $u \equiv [q(\bar{t}_n) = S]$. By definition of buggy vertex, all the immediate descendants of u are valid. Since vertex u is invalid, by Proposition 4, the relation q cannot be well-defined.
2. Suppose that G contains a buggy circuit $C \equiv u_1, \dots, u_n$ with $u_n = u_1$ and each u_i of the form $[A_i = S_i]$ for $i = 1 \dots n-1$. We consider two possibilities:
   (a) At least one of the vertices in the circuit contains a wrong answer. Let S be the set of the wrong atom instances contained in the circuit:

$$S = \{B \in S_i \wedge B \notin \mathcal{I} \mid \text{ for some } 1 \leq i < n\}$$

   Obviously, $S \subseteq \mathcal{M}$ and $S \cap \mathcal{I} = \emptyset$. Now, we consider a stratification $\{P_1, \dots, P_k\}$ of the program P and the sequence of Herbrand interpretations starting from $\emptyset$ and ending in $\mathcal{M}$ defined in item 5 of Proposition 1. We single out the first interpretation in this sequence including some element of S. Such interpretation must be of the form $T_{P_i}(I)$, with I the previous interpretation in the sequence and $1 \leq i \leq k$. Let $p(\bar{s}_n)$ be an element of $T_{P_i}(I) \cap S$. By definition of T_P, there exists a substitution θ and a program rule $(p(\bar{t}_n) :- l_1, \dots, l_m) \in P$ s.t. $p(\bar{s}_n) = p(\bar{t}_n)\theta$ and that $I \models l_i\theta$ for every $i = 1 \dots m$. By Proposition 3, each l_i, $i = 1 \dots m$, has some associated vertex V' successor of V in the CG with V' of the form $[atom(l_i)\sigma = \dots]$ with σ more general than θ. We distinguish two possibilities:

- V' is out of the circuit. Then, by the definition of buggy circuit V' is valid w.r.t. $\mathcal{I}$, which means all the instances of $l_i\theta$ are also valid w.r.t. $\mathcal{I}$. This is true independently of whether l_i is positive or negative because the validity of the answer for a query implies the validity of the answer for its negation in our setting.
- V' is in the circuit. Then, l_i is positive due to Lemma 1, and by construction, all the instances of $l_i\theta$ included in $\mathcal{I}$ are valid w.r.t. $\mathcal{I}$.

In any case, $\mathcal{I} \models l_i\theta$ for every $i = 1 \ldots m$ but $p(\bar{t}_n)\theta \notin \mathcal{I}$ and hence p is an *incorrect relation*.

(b) If none of the vertices in the buggy circuit contains a wrong answer, then every vertex contains a missing answer.
Put

$$S = \{A_i\sigma \in \mathcal{I}, A_i\sigma \notin S_i \mid \text{for some } 1 \leq i < k\}$$

i.e., S is the set of missing instances in the circuit. Next, we check that S is an uncovered set of atoms, which means that the relations in the buggy circuit form an incomplete set of relations. Let $A_j\sigma \in S$ be an atom of S with $1 \leq j \leq k$, $(p(\bar{t}_n) :- l_1, \ldots, l_m) \in P$ be a program rule, and $\theta \in Subst$ such that:
- $p(\bar{t}_n)\theta = A_j\sigma$,
- $\mathcal{I} \models l_i\theta$ for $i = 1 \ldots m$

There must exist at least one $l_i\theta \notin \mathcal{M}$, $1 \leq i \leq m$, otherwise $A_j\sigma$ would be in $\mathcal{M}$. Let r be the least index, $1 \leq r \leq m$, s.t. $l_r\theta \notin \mathcal{M}$. By Lemma 2, there is a successor of $[A_j = S_j]$ in G of the form $w = [l_r\theta' = S_r]$ with $\theta \geq \theta'$. Then, $l_r\theta$ is a missing answer for w, i.e., it is an invalid vertex (it is easy to prove that, if $l\theta$ has a missing answer, then $l\theta'$ has a missing answer for every θ' s.t. $\theta \geq \theta'$). This implies that $w \in C$, and hence l_r is a positive literal (by Lemma 1), $l_i\theta \in S$, and S is uncovered. □

After the soundness result, it remains to prove that the technique is complete:

Proposition 7 (Completeness). *Let P be a Datalog program and Q be a query with answer $Q_{\mathcal{M}}$ unexpected. Then, the computation graph G for Q w.r.t. P contains either a buggy node or a buggy circuit.*

Proof. By the construction of the computation graph, G contains a vertex for $[atom(Q) = atom(Q)_{\mathcal{M}}]$. If Q is positive, then $Q = atom(Q)$ and the vertex is of the form $[Q = Q_{\mathcal{M}}]$. Then, by hypothesis, $Q_{\mathcal{M}}$ is unexpected, and therefore the vertex is invalid. If Q is negative and it has an unexpected answer, it is straightforward to check that $atom(Q)$ also produces an unexpected answer and hence $[atom(Q) = atom(Q)_{\mathcal{M}}]$ is also invalid. Then, the result is a direct consequence of Proposition 5.

5 Implementation

The theoretical ideas explained so far have been implemented in a debugger included as part of the Datalog system DES [13]. The *CG* is built after the user

has detected some unexpected answer. The values $(Q, Q_{\mathcal{M}_A})$ are stored along the computation and can be accessed afterwards without repeating the computation, thus increasing the efficiency of the graph construction.

A novelty of our approach is that it allows the user to choose working either at clause level or at predicate level, depending on the grade of precision that the user needs, and its knowledge of the intended interpretation $\mathcal{I}$. At predicate level, the debugger is able to find a buggy relation or an incomplete set of relations. At clause level, the debugger can provide additional information, namely the rule which is the cause of error.

For instance, next is the debugging session at predicate level for the query `planet(X)` w.r.t. our running example:

```
DES> /debug planet(X) p

Info: Starting debugger...

Is orbits(sun,sun) = {}  valid(v)/invalid(n)/abort(a) [v]? v
Is orbits(earth,Y) = {orbits(earth,sun)}
                      valid(v)/invalid(n)/abort(a) [v]? v
Is intermediate(earth,sun) = {intermediate(earth,sun)}
                      valid(v)/invalid(n)/abort(a) [v]? n
Is orbits(sun,Y) = {}  valid(v)/invalid(n)/abort(a) [v]? v
Is orbits(X,sun) = {orbits(earth,sun),orbits(moon,sun)}
                          valid(v)/invalid(n)/abort(a) [v]? v

Error in relation: intermediate/2
Witness query:
      intermediate(earth,sun) = {intermediate(earth,sun)}
```

The first question asks whether the query `orbits(sun,sun)` is expected to fail, i.e., it yields no answer. This is the case because we do not consider the sun orbiting around itself. The answer to the second question is also `valid` because the earth orbits only the sun in our intended model. But the answer to the next question is `invalid`, since the query `intermediate(earth,sun)` should fail because the earth orbits directly the sun. The next two answers are `valid`, and with this information the debugger determines that there is a buggy node in the CG corresponding to the relation `intermediate/2`, which is therefore buggy. The *witness query* shows the instance that contains the unexpected instance. This information can be useful for locating the bug.

In order to minimize the number of questions asked to the user, the tool relies on a navigation strategy similar to the *divide & query* presented in [12] for deciding which vertex is selected at each step. In other paradigms it has been shown that this strategy requires an average of $\log_2 n$ questions to find the bug [19], with n the number of nodes in the computation tree. Our experiments confirms that this is also the case when the CGs are in fact trees, i.e., they do not contain cycles, which occurs very often. In the case of graphs containing cycles the results also show this tendency, although a more extensive number of experiments is still needed.

6 Conclusions and Future Work

We have applied declarative debugging to Datalog programs. The debugger detects incorrect fragments of code starting from an unexpected answer. In order to find the bug, the tool requires the help of the user as an external oracle answering questions about the validity of the results obtained for some subqueries. We have proved formally the completeness and soundness of the technique, thus proposing a solid foundations for the debugging of Datalog programs. During the theoretical study, we have found that the traditional errors considered usually in logic programming are not enough in the case of Datalog where a new kind of error, the incomplete sets of predicates, can occur.

The theoretical ideas have been set in practice by developing a declarative debugger for the Datalog system DES. The debugger allows diagnosing both missing and wrong answers, which constitute all the possible errors symptoms of a Datalog program. Although a more extensive workbench is needed, the preliminary experiments are encouraging about the usability of the tool. The debugger allows to detect readily errors which otherwise would take considerable time. This is particularly important for the DES system, which has been developed with educational purposes. By using the debugger, the students can find the errors in a program by considering only its declarative meaning and disregarding operational issues.

From the point of view of efficiency, the results are also quite satisfactory. The particular characteristics of DES make all the information necessary for producing the graph available after each computation. The answers to each subquery, therefore, are not actually computed in order to build the graph but simply pointed to. This greatly speeds up the graph construction and keeps small the size of the graph even for large computations.

As future work, we consider the possibility of allowing more elaborated answers from the user. For instance, indicating that a vertex is not only invalid but also that it contains a wrong answer. The identification of such an answer can greatly reduce the number of questions. Another task is to develop and compare different navigation strategies for minimizing the number of questions needed for finding the bug.

References

1. Ramakrishnan, R., Ullman, J.: A survey of research on Deductive Databases. The Journal of Logic Programming 23(2), 125–149 (1993)
2. Beeri, C., Ramakrishnan, R.: On the power of magic. In: Proceedings of the Sixth ACM Symposium on Principles of Database Systems, pp. 269–284 (1987)
3. Dietrich, S.W.: Extension tables: Memo relations in logic programming. In: SLP, pp. 264–272 (1987)
4. Arora, T., Ramakrishnan, R., Roth, W.G., Seshadri, P., Srivastava, D.: Explaining program execution in deductive systems. In: Deductive and Object-Oriented Databases, pp. 101–119 (1993)
5. Wieland, C.: Two explanation facilities for the deductive database management system DeDEx. In: Kangassalo, H. (ed.) ER 1990, pp. 189–203, ER Institute (1990)

6. Specht, G.: Generating explanation trees even for negations in deductive database systems. In: Proceedings of the 5th Workshop on Logic Programming Environments, Vancouver, Canada (1993)
7. Russo, F., Sancassani, M.: A declarative debugging environment for Datalog. In: Proceedings of the First Russian Conference on Logic Programming, pp. 433–441. Springer, London (1992)
8. Baral, C.: Knowledge representation, reasoning, and declarative problem solving. Cambridge University Press, Cambridge (2003)
9. Syrjänen, T.: Debugging inconsistent answer set programs. In: Proceedings of the 11th International Workshop on Non-Monotonic Reasoning, Lake District, UK, pp. 77–84 (May 2006)
10. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging ASP Programs by Means of ASP. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 31–43. Springer, Heidelberg (2007)
11. Caballero, R., García-Ruiz, Y., Sáenz-Pérez, F.: A new proposal for debugging datalog programs. In: 16th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2007) (June 2007)
12. Shapiro, E.: Algorithmic Program Debugging. In: ACM Distiguished Dissertation. MIT Press, Cambridge (1982)
13. Sáenz-Pérez, F.: Datalog Educational System. User's Manual. Technical Report 139-04, Facultad de Informática, Universidad Complutense de Madrid (2004), http://des.sourceforge.net/
14. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Foundations of deductive databases and logic programming, pp. 89–148. Morgan Kaufmann Publishers Inc., San Francisco (1988)
15. Ullman, J.: Database and Knowledge-Base Systems Vols. I (Classical Database Systems) and II (The New Technologies). Computer Science Press (1995)
16. Chandra, A.K., Harel, D.: Horn clauses queries and generalizations. J. Log. Program. 2(1), 1–15 (1985)
17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K. (eds.) Proceedings of the Fifth International Conference on Logic Programming, pp. 1070–1080. MIT Press, Cambridge (1988)
18. Naish, L.: A Declarative Debugging Scheme. Journal of Functional and Logic Programming 3 (1997)
19. Caballero, R.: A declarative debugger of incorrect answers for constraint functional-logic programs. In: WCFLP 2005: Proceedings of the 2005 ACM SIGPLAN workshop on Curry and functional logic programming, pp. 8–13. ACM Press, New York (2005)

Semantic Bijectivity and the Uniqueness of Constant-Complement Updates in the Relational Context

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. Within the context of the relational model, a general technique for establishing that the translation of a view update defined by constant complement is independent of the choice of complement is presented. In contrast to previous results, the uniqueness is not limited to order-based updates (those constructed from insertions and deletions), nor is it limited to those well-behaved complements which define closed update strategies. Rather, the approach is based upon optimizing the change of information in the main schema which the view update entails. The only requirement is that the view and its complement together possess a property called *semantic bijectivity* relative to the information measure. It is furthermore established that a very wide range of views have this property. This results formalizes the intuition, long observed in examples, that it is difficult to find different complements which define distinct but reasonable update strategies.

1 Introduction

It has long been recognized that there is no ideal solution to the problem of supporting updates to views of database schemata. Rather, all solutions involve compromise of some sort. At the most conservative end of the spectrum lies the constant-complement strategy, introduced by Bancilhon and Spyratos more than a quarter-century ago [1]. It has recently seen renewed interest, both on the theoretical front [2] [3] and as a framework for applications [4], to no small extent because it is precisely the strategy which avoids all so-called update anomalies [3, Sec. 1].

The idea behind the constant-complement strategy is simple. Let $\mathbf{D}$ be a database schema, with $\mathsf{LDB}(\mathbf{D})$ denoting its set of legal states. An update on $\mathbf{D}$ is just a pair $(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$ in which M_1 is the current state and M_2 is the new state after the update. A view of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is the view schema and $\gamma : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V})$ is the view mapping. Since a view is window on the main schema, its state must always be determined by that of the main schema $\mathbf{D}$; hence, γ is always taken to be surjective. Let $M_1 \in \mathsf{LDB}(\mathbf{D})$

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 160–179, 2008.

be the current state of the main schema $\mathbf{D}$, so that $\gamma(M_1)$ is the current state of the view Γ. A translation of the update request $(\gamma(M_1), N_2) \in \mathsf{LDB}(\mathbf{V}) \times \mathsf{LDB}(\mathbf{V})$ (relative to M_1) is an update (M_1, M_2) on $\mathbf{D}$ with the property that $\gamma(M_2) = N_2$. In general, there are many such translations. However, now let $\Gamma' = (\mathbf{V}', \gamma')$ be a second view of $\mathbf{D}$. The decomposition mapping for $\{\Gamma, \Gamma'\}$ is $\gamma \coprod \gamma' : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V}) \coprod \mathsf{LDB}(\mathbf{V}')$ given on elements by $M \mapsto (\gamma(M) \uplus \gamma'(M))$. Here $\uplus$ is the disjoint union operator; thus, the decomposition mapping retains the state of each constituent view. (See Definition 3.4 for details.) The view Γ' is called a complement of Γ, and $\{\Gamma, \Gamma'\}$ is called a complementary pair, if this decomposition mapping is injective (or lossless), so that the state of $\mathbf{D}$ is recoverable from the combined states of the two views. A translation (M_1, M_2) of the proposed view update $(\gamma(M_1), N_2)$ to Γ has constant complement Γ' if $\gamma'(M_1) = \gamma'(M_2)$. If $\{\Gamma, \Gamma'\}$ forms a complementary pair, then it is easy to see that there can be at most one translation of $(\gamma(M_1), N_2)$ which has constant complement Γ', since there can be at most one $M_2 \in \mathsf{LDB}(\mathbf{D})$ with the property that $(\gamma \coprod \gamma')(M_2) = (N_2, \gamma'(M_1))$. Thus, upon fixing a complement Γ' to Γ, translations of view updates become unique.

There is a very desirable feature which limits the choice of complement Γ'. In general, the user of the view Γ will not know the precise state M_1 of the main schema; rather, only the image $\gamma(M_1)$ will be known within the context of Γ. Ideally, whether or not the view update $(\gamma(M_1), N_2)$ is allowed should depend only upon $\gamma(M_1)$ and N_2, and not upon M_1 itself. In other words, given $M_1' \in \mathsf{LDB}(\mathbf{D})$ with the property that $\gamma(M_1) = \gamma(M_1')$, the view update $(\gamma(M_1), N_2)$ should either be allowed for both M_1 and M_1', or else for neither. In [3, 2.14], it is shown that this condition recaptured by requiring that the pair $\{\Gamma, \Gamma'\}$ be meet complementary, in the sense that the congruences on $\mathsf{LDB}(\mathbf{D})$ defined by γ and γ' commute.

Even upon limiting attention to commuting congruences, there is a further complication surrounding the constant-complement approach; namely, the translation of a view update to the main schema is dependent upon the choice of complement, and except in the simplest of situations, there are many possible complements. In contrast to this theoretical result, in practice it is often clear that there is a natural translation of view updates to the main schema which is supported by the "obvious" complement, with other more contrived translations supported by equally contrived complements. In [3], it is argued that there is a key additional property which "good" views have; namely, that they are monotonic with respect to the natural order on states. In the context of the relational algebra, this amounts to excluding negation. With this additional condition enforced, it has been shown that the translation of a view update to the main schema is independent of the choice of complement [3, 4.3], with the limitation that the updates themselves are *order realizable*; that is, realizable as sequences of legal insertions and deletions.

In this paper, the issue of relaxing this limitation to order-based updates is addressed with the context of the classical relational model. Rather than using a syntactic notion of monotonicity based upon the order structure of database

states, a semantic formulation, which characterizes the *information content* of a database state in terms of the set of sentences in a particular family which it satisfies, is employed. The decomposition mapping is then required to be not only bijective in the ordinary sense but also a *semantic bijection*, in the sense that it defines a bijection between (semantic equivalence classes of) the sentences which define the information content. Under this requirement, it is shown that constant-complement translations of all view updates, order based or not, are always information optimal, and hence unique and independent of the choice of complement. It is furthermore shown that when the main schema is constrained by a wide range of classical database dependencies, and the view mappings are SPJ-mappings, that is, conjunctive queries, these conditions are always satisfied. Interestingly, this result is not limited to meet-complementary pairs in the sense of [3, 2.12]; that is, pairs of view which define well-behaved families of updates. Rather, it applies to any pair of complementary views whose decomposition mapping is a semantic bijection, and any update defined via constant-complement within that context.

This work is based upon the notions of information content and optimal translations which were introduced in [5]. In that work, it is argued that view updates should be *information optimal*, in that the correct reflection should entail the least information change over all possibilities. However, the information measure in that case allows for the equivalence of updates which are isomorphic in a certain sense. This work applies that philosophy under the additional constraint that the update strategy should be defined by constant complement, with the reflected updates to the main schema truly unique, and not just isomorphic.

Although the results are vastly different, some of the background material of this paper is similar or identical to that of [5]. Specifically, much of Section 2, as well as some content of Definition 3.2, Definition 4.2, Definition 4.3, Definition 4.17 and Definition 4.19 is adapted from [5], although numerous changes in detail and often simplifications have been made to accommodate the specific needs of this work.

2 The Relational Model

The results of this paper are formulated within the relational model, and familiarity with its standard notions, as presented in references such as [6] and [7], is assumed. Nevertheless, there are aspects which must be formulated with particular care. Most important are the need to take all relational schemata over the same domain, with the same constant symbols, and the need to express databases themselves as sets of ground atoms. For this reason, the key features which are unique to this formulation are presented in this section.

Definition 2.1 (Relational contexts and constant interpretations). A relational context contains the logical information which is shared amongst the schemata and database mappings. Formally, a relational context $\mathcal{D}$ consists of a finite nonempty set $\mathcal{A}_\mathcal{D}$ of attribute names, a countable set $\mathsf{Vars}(\mathcal{D})$ of variables, and for each $A \in \mathcal{A}_\mathcal{D}$, an at-most-countable set $\mathsf{Const}_\mathcal{D}(A)$ of constant symbols,

with $\mathsf{Const}(\mathcal{D}) = \bigcup\{\mathsf{Const}_{\mathcal{D}}(A) \mid A \in \mathcal{A}_{\mathcal{D}}\}$. The variables in $\mathsf{Vars}(\mathcal{D})$ are further partitioned into two disjoint sets; a countable set $\mathsf{GenVars}(\mathcal{D}) = \{x_0, x_1, x_2, \ldots\}$ of *general variables*, and special $\mathcal{A}_{\mathcal{D}}$-indexed set $\mathsf{AttrVars}(\mathcal{D}) = \{x_A \mid A \in \mathcal{A}_{\mathcal{D}}\}$ of *attribute variables*. The latter is used in the definition of interpretation mappings; see Definition 2.6 for details.

Databases are represented as ground atoms, as elaborated in Definition 2.2 below. Therefore, it is necessary that each domain element be bound to a unique constant symbol. Formally, a *constant interpretation* for the relational context $\mathcal{D}$ is a pair $\mathcal{I} = (\mathsf{Dom}_{\mathcal{I}}, \mathsf{IntFn}_{\mathcal{I}})$ in which $\mathsf{Dom}_{\mathcal{I}}$ is a countably infinite set, called the *domain* of $\mathcal{I}$, and $\mathsf{IntFn}_{\mathcal{I}} : \mathsf{Const}(\mathcal{D}) \to \mathsf{Dom}_{\mathcal{I}}$ is a bijective function, called the *interpretation function* of $\mathcal{I}$. This effectively stipulates the following two well-known conditions [8, p. 120]:

Domain closure: $(\forall x)(\bigvee_{a \in \mathsf{Const}(\mathcal{D})} x = a)$ $(\mathsf{DCA}(\mathcal{D}))$
Unique naming: $(\neg(a = b))$ for distinct $a, b \in \mathsf{Const}(\mathcal{D})$ $(\mathsf{UNA}(\mathcal{D}))$

Since there are countably many constant symbols, the domain-closure axiom is not a finite disjunction. This is not a problem however, since it is never used in a context in which a first-order constraint is necessary. Because the assignment of domain values to constants is fixed, it is not necessary to verify independently that it holds.

As a notational convention, from this point on, unless stated otherwise, fix a relational context $\mathcal{D}$ and a constant interpretation $\mathcal{I} = (\mathsf{Dom}_{\mathcal{I}}, \mathsf{IntFn}_{\mathcal{I}})$ for it.

Definition 2.2 (Tuples and databases). An *unconstrained relational schema* over $(\mathcal{D}, \mathcal{I})$ is a pair $\mathbf{D} = (\mathsf{Rels}(\mathbf{D}), \mathsf{Ar}_{\mathbf{D}})$ in which $\mathsf{Rels}(\mathbf{D})$ is finite set of relational symbols and $\mathsf{Ar}_{\mathbf{D}} : \mathsf{Rels}(\mathbf{D}) \to \mathbf{2}^{\mathcal{A}_{\mathcal{D}}}$ a function which assigns an *arity*, a set of distinct attributes from $\mathcal{A}_{\mathcal{D}}$, to each $R \in \mathsf{Rels}(\mathbf{D})$.

A *ground* R*-atom* is a function $t : \mathsf{Ar}_{\mathbf{D}}(R) \to \mathsf{Const}(\mathcal{D})$ with the property that $t[A] \in \mathsf{Const}_{\mathcal{D}}(A)$. The set of all ground R-atoms is denoted $\mathsf{GrAtoms}(R, \mathbf{D})$. A *ground* $\mathbf{D}$*-atom* is a ground R-atom for some $R \in \mathsf{Rels}(\mathbf{D})$, with the set of all ground $\mathbf{D}$-atoms denoted $\mathsf{GrAtoms}(\mathbf{D})$. An *atom database for* $\mathbf{D}$ is a finite subset of $\mathsf{GrAtoms}(\mathbf{D})$, with the set of all atom databases for $\mathbf{D}$ denoted $\mathsf{DB}(\mathbf{D})$.

An R*-atom* t is defined similarly, except that $t[A]$ is not required to be a constant; rather, $t[A] \in \mathsf{Const}_{\mathcal{D}}(A) \cup \mathsf{GenVars}(\mathcal{D}) \cup \{x_A\}$. The $\mathbf{D}$*-atoms* and the set $\mathsf{Atoms}(\mathbf{D})$ are defined in the obvious way.

It is convenient to be able to recover the associated relation name from a tuple, and so *tagging* is employed, in which tuples are marked with the relation name. Formally, this is accomplished by introducing a new attribute $\mathsf{RName} \notin \mathcal{A}_{\mathcal{D}}$, and then regarding a ground R-atom not as a function t just on $\mathsf{Ar}_{\mathbf{D}}(R)$ but rather as one on $\{\mathsf{RName}\} \cup \mathsf{Ar}_{\mathbf{D}}(R)$ with the property that $t[\mathsf{RName}] = R$. Tagging of R-atoms will be used from this point on throughout the paper. Nevertheless, in writing such atoms, the more conventional notation $R(a_1, a_2, \ldots, a_n)$ will be used in lieu of the technically more correct $(R, a_1, a_2, \ldots, a_n)$, although tags will be used in formal constructions.

Definition 2.3 (Formulas and constraint classes). The first-order language associated with the relational schema $\mathbf{D}$ is defined in the natural way;

however, it is useful to introduce some notation which identifies particular sets of formulas. Define $\mathsf{WFF}(\mathbf{D})$ to be the set of all well-formed first-order formulas with equality in the language whose set of relational symbols is $\mathsf{Rels}(\mathbf{D})$, whose set of constant symbols is $\mathsf{Const}(\mathcal{D})$, and which contains no non-nullary function symbols. The variables are those of $\mathcal{D}$; these formulas are typed only to the extent that for $A \in \mathcal{A}_{\mathcal{D}} D$, the variable x_A may only occur in a $\mathbf{D}$-atom in a position associated with attribute A. In other words, the conditions for $\mathbf{D}$-atoms identified in Definition 2.2 are enforced. $\mathsf{WFS}(\mathbf{D})$ denotes the subset of $\mathsf{WFF}(\mathbf{D})$ consisting of sentences; that is, formulas with no free variables.

A *constraint class* $\mathcal{C}$ identifies a subset of $\mathsf{WFF}(\mathbf{D})$, denoted $\mathsf{WFF}(\mathbf{D}, \mathcal{C})$. Of particular interest in this work are $\exists\neq$, $\exists+$, $\exists\wedge+$, Atoms, and $\mathbf{1}$, defined as follows.

- $\mathsf{WFF}(\mathbf{D}, \exists\neq)$ is the subset of $\mathsf{WFF}(\mathbf{D})$ consisting of those formulas in which only existential quantification is allowed, and in which negation (explicit or implicit) occurs only at the level of equality atoms. More precisely, negation may only occur in the form $\neg(\tau_1 = \tau_2)$, with τ_1 and τ_2 terms. Negation of other atoms, such as in $(\exists x_1)(\exists x_2)(R(x) \wedge (\neg S(x))$, is prohibited.
- $\mathsf{WFF}(\mathbf{D}, \exists+)$ is the subset of $\mathsf{WFF}(\mathbf{D}, \exists\neq)$ in which no negation at all is allowed.
- $\mathsf{WFF}(\mathbf{D}, \exists\wedge+)$ is the subset of $\mathsf{WFF}(\mathbf{D}, \exists+)$ in which disjunction is also disallowed, so that the only logical connective which is allowed is conjunction. These formulas define the so-called *conjunctive queries* [9, Sec. 4.2].
- $\mathsf{WFF}(\mathbf{D}, \mathsf{Atoms})$ is just $\mathsf{Atoms}(\mathbf{D})$.
- $\mathsf{WFF}(\mathbf{D}, \mathbf{1})$ is shorthand for $\mathsf{WFF}(\mathbf{D})$.

In each case, the corresponding set $\mathsf{WFS}(\mathbf{D}, \mathcal{C})$ of sentences is defined in the obvious way. In particular, note that $\mathsf{WFS}(\mathbf{D}, \mathsf{Atoms}) = \mathsf{GrAtoms}(\mathbf{D})$.

Definition 2.4 (Atomic models). Even though databases are represented as sets of ground atoms, and not as interpretations in the usual logical sense, it is still essential to have an appropriate notion of model for a given sentence. This is relatively straightforward; a model for a sentence φ is a database which is consistent with both φ and the unique-naming axioms. There is one complication, however. In representing a database as a set of $\mathbf{D}$-atoms, the closed-world assumption is implicit. On the other hand, to express what it means for such a representation to satisfy an arbitrary sentence in $\mathsf{WFS}(\mathbf{D})$, it is necessary to state explicitly which atoms are not true as well. Formally, for $M \in \mathsf{DB}(\mathbf{D})$, define the *diagram* of M to be $\mathsf{Diagram}_{\mathbf{D}}(M) = M \cup \{\neg t \mid t \in \mathsf{GrAtoms}(\mathbf{D}) \setminus M\}$. Now, say that $M \in \mathsf{DB}(\mathbf{D})$ is an *atomic $\mathcal{I}$-model* of $\varphi \in \mathsf{WFS}(\mathbf{D})$ if $\mathsf{Diagram}_{\mathbf{D}}(M) \cup \{\varphi\} \cup \mathsf{UNA}(\mathcal{D})$ is consistent. $\mathsf{AtMod}_{\mathcal{I}}(\varphi)$ denotes the set of all atomic $\mathcal{I}$-models of φ, with $\mathsf{AtMod}_{\mathcal{I}}(\Phi) = \bigcap\{\mathsf{AtMod}_{\mathcal{I}}(\varphi) \mid \varphi \in \Phi\}$ for $\Phi \subseteq \mathsf{WFS}(\mathbf{D})$.

Definition 2.5 (Schemata with constraints and constrained databases). To obtain full relational schemata, constraints are added to the unconstrained schemata of Definition 2.2. Formally, a *relational schema* over $(\mathcal{D}, \mathcal{I})$ is a triple $\mathbf{D} = (\mathsf{Rels}(\mathbf{D}), \mathsf{Ar}_{\mathbf{D}}, \mathsf{Constr}(\mathbf{D}))$ in which $(\mathsf{Rels}(\mathbf{D}), \mathsf{Ar}_{\mathbf{D}})$ is an

unconstrained relational schema over $(\mathcal{D}, \mathcal{I})$ and $\mathsf{Constr}(\mathbf{D}) \subseteq \mathsf{WFS}(\mathbf{D})$ is the set of *dependencies* or *constraints* of $\mathbf{D}$.

Define the *legal* (or *constrained*) *databases* $\mathsf{LDB}(\mathbf{D})$ of $\mathbf{D}$ to be $\mathsf{AtMod}_{\mathcal{I}}(\mathsf{Constr}(\mathbf{D}))$.

Define the equivalence relation $\equiv_{\mathbf{D}}$ on $\mathsf{WFS}(\mathbf{D})$ by $\varphi_1 \equiv_{\mathbf{D}} \varphi_2$ iff $\mathsf{AtMod}_{\mathcal{I}}(\varphi_1) \cap \mathsf{LDB}(\mathbf{D}) = \mathsf{AtMod}_{\mathcal{I}}(\varphi_2) \cap \mathsf{LDB}(\mathbf{D})$ or, equivalently, $\mathsf{AtMod}_{\mathcal{I}}(\{\varphi_1\} \cup \mathsf{Constr}(\mathbf{D})) = \mathsf{AtMod}_{\mathcal{I}}(\{\varphi_2\} \cup \mathsf{Constr}(\mathbf{D}))$. Thus, $\equiv_{\mathbf{D}}$ identifies sentences which have identical truth values on all $M \in \mathsf{LDB}(\mathbf{D})$. The equivalence class of φ_1 under $\equiv_{\mathbf{D}}$ is denoted $[\varphi_1]_{\equiv_{\mathbf{D}}}$.

Definition 2.6 (Database morphisms and views). Let $\mathbf{D}_1$ and $\mathbf{D}_2$ be relational schemata over $(\mathcal{D}, \mathcal{I})$. There are two fundamental ways to represent a database morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ in the relational context. On the one hand, such a morphism may be represented as a function $f : \mathsf{DB}(\mathbf{D}_1) \to \mathsf{DB}(\mathbf{D}_2)$, using expressions from the relational algebra. On the other hand, by providing an interpretation formula $f^R \in \mathsf{WFF}(\mathbf{D}_1)$ for each $R \in \mathsf{Rels}(\mathbf{D}_2)$, the morphism may be represented using the relational calculus [10]. The equivalence of these two representations is one of the classical results of relational database theory [6, Sec. 2.4-2.6]. The interpretation formulation is taken as the basic one in this work. Formally, given $R \in \mathsf{Rels}(\mathbf{D}_2)$, an *interpretation* for R into $\mathbf{D}_1$ is a $\varphi \in \mathsf{WFF}(\mathbf{D})$ in which precisely the variables $\{x_A \mid A \in \mathsf{Ar}_{\mathbf{D}}(R)\}$ are free, with x_A is used to mark the position in the formula which is bound to attribute A. The set of all interpretations of R into $\mathbf{D}_1$ is denoted $\mathsf{Interp}(R, \mathbf{D}_1)$. A *syntactic morphism* $f : \mathbf{D}_1 \to \mathbf{D}_2$ is a family $f = \{f^R \mid R \in \mathsf{Rels}(\mathbf{D}_2) \text{ and } f^R \in \mathsf{Interp}(R, \mathbf{D}_1)\}$.

Let $t \in \mathsf{Atoms}(R, \mathbf{D}_2)$. The *substitution* of t into f, denoted $\mathsf{Subst}\langle f, t\rangle$, is the formula in $\mathsf{WFF}(\mathbf{D}_1)$ obtained by substituting $t[A]$ for x_A, for each $A \in \mathsf{Ar}_{\mathbf{D}}(R)$. Note that If t is a ground atom, then $\mathsf{Subst}\langle f, t\rangle \in \mathsf{WFS}(\mathbf{D}_1)$.

For $M \in \mathsf{DB}(\mathbf{D}_1)$, define $f(M) = \{t \in \mathsf{GrAtoms}(\mathbf{D}_2) \mid M \in \mathsf{AtMod}_{\mathcal{I}}(\mathsf{Subst}\langle f, t\rangle)\}$. f is called an LDB-*morphism* if it maps legal databases to legal databases; formally, an LDB-morphism has the property that $f(M) \in \mathsf{LDB}(\mathbf{D}_2)$ for each $M \in \mathsf{LDB}(\mathbf{D}_1)$. When no qualification is given, *database morphism* will always mean LDB-morphism.

Let $\mathbf{D}$ be a relational schema over $(\mathcal{D}, \mathcal{I})$. A *(relational) view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is a relational schema over $(\mathcal{D}, \mathcal{I})$ and $\gamma : \mathbf{D} \to \mathbf{V}$ is an LDB-morphism which is furthermore LDB-*surjective* in the sense that for every $N \in \mathsf{LDB}(\mathbf{V})$, there is an $M \in \mathsf{LDB}(\mathbf{D})$ with $\gamma(M) = N$. Surjectivity is required because the state of the view must always be determined by the state of the main schema $\mathbf{D}$.

3 Updates and View Complements

In this section, some basic definitions and notation regarding updates and their reflections are presented, as are the key ideas surrounding the constant-complement update strategy.

Notation 3.1. Throughout this section, take $\mathbf{D}$ to be a relational schema over $(\mathcal{D}, \mathcal{I})$.

Definition 3.2 (Updates and reflections). Let $\Gamma = (\mathbf{V}, \gamma)$ be a view on $\mathbf{D}$. An *update* on $\mathbf{D}$ is a pair $(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$. M_1 is the current state, and M_2 the new state. To describe the situation surrounding an update request on Γ, it is sufficient to specify the current state M_1 of the main schema and the desired new state N_2 of the view schema $\mathbf{V}$. The current state of the view can be computed as $\gamma(M_1)$; it is only the new state M_2 of the main schema (subject to $N_2 = \gamma(M_2)$) which must be obtained from an update strategy. Formally, an *update request* from Γ to $\mathbf{D}$ is a pair (M_1, N_2) in which $M_1 \in \mathsf{LDB}(\mathbf{D})$ (the old state of the main schema) and $N_2 \in \mathsf{LDB}(\mathbf{V})$ (the new state of the view schema). A *realization* of (M_1, N_2) along Γ is an update (M_1, M_2) on $\mathbf{D}$ with the property that $\gamma(M_2) = N_2$. The update (M_1, M_2) is called a *reflection* (or *translation*) of the view update $(\gamma(M_1), N_2)$. The set of all realizations of (M_1, N_2) along Γ is denoted $\mathsf{UpdRealiz}\langle M_1, N_2, \Gamma\rangle$.

Notation 3.3 (Disjoint union). In the construction of complementary views, the disjoint union of two sets will be used to construct the coproduct of views. As the symbol $\coprod$ will be reserved to denote a formal coproduct, $\uplus$ will be used to represent the disjoint union of two sets. Thus, $A \uplus B$ is just a "tagged" version of $A \cup B$, in which it is possible to determine from which of the two sets an element arose. Note that it is possible for there to be two instances of a given element x in $A \uplus B$, one tagged with A and the other tagged with B.

Definition 3.4 (The coproduct of two views). The coproduct of two views is the natural one which arises when complementation is considered. (The terminology of and notation for coproduct is used because this construction is a true coproduct in the categorical sense [11, §18]). Basically, the set of relations of the coproduct schema is the disjoint union of those of the two component schemata, with each such relation retaining its interpretation function. The databases are defined similarly via disjoint union. (Recall that tuples are tagged (see Definition 2.2), so a database for a schema consists of just one big set of tuples.) Formally, let $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ and $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ be views of $\mathbf{D}$. The *coproduct* of Γ_1 and Γ_2 is the view $\Gamma_1 \coprod \Gamma_2 = (\mathbf{V}_1 \coprod \mathbf{V}_2, \gamma_1 \coprod \gamma_2)$, defined as follows.

(a) $\mathsf{Rels}(\mathbf{V}_1 \coprod \mathbf{V}_2) = \mathsf{Rels}(\mathbf{V}_1) \uplus \mathsf{Rels}(\mathbf{V}_2)$.
(b) For $i \in \{1, 2\}$ and $R \in \mathsf{Rels}(\mathbf{V}_i)$, $(\gamma_1 \coprod \gamma_2)^R = \gamma_i^R$.
(c) $\mathsf{LDB}(\mathbf{V}_1 \coprod \mathbf{V}_2) = \{\gamma_1(M) \uplus \gamma_2(M) \mid M \in \mathsf{LDB}(\mathbf{D})\}$.
(d) $\mathsf{Constr}(\mathbf{V}_1 \coprod \mathbf{V}_2)$ is the set of all first-order sentences which define the constraints on $\mathsf{LDB}(\mathbf{V}_1 \coprod \mathbf{V}_2)$.

Note that, in view of (c), $\gamma_1 \coprod \gamma_2$ is surjective by construction. Hence, there is no question that $\gamma_1 \coprod \gamma_2$ is a view.

In general, there is no straightforward representation for $\mathsf{Constr}(\mathbf{V}_1 \coprod \mathbf{V}_2)$, the set of constraints of this coproduct. Nevertheless, it can be shown that it has a representation as a set of first-order sentences [12, Ch. 26]. Since this representation is not central to the theme of this paper, it will not be elaborated further.

It should perhaps also be noted that, strictly speaking, the notation $\mathbf{V}_1 \coprod \mathbf{V}_2$ is incomplete, since this product depends not only upon $\mathbf{V}_1$ and $\mathbf{V}_2$, but upon

the view morphisms γ_1 and γ_2 as well. However, since no confusion can result, a more accurate but correspondingly cumbersome notation will not be introduced.

Definition 3.5 (Constant-complement realizations and complementary pairs). Although the ideas surrounding constant-complement update are well known [1] [3], it is important to formalize them within the current context. Let Γ_1 and Γ_2 be views of $\mathbf{D}$.

(a) $\{\Gamma_1, \Gamma_2\}$ forms a *complementary pair* if the underlying function $\gamma_1 \coprod \gamma_2 : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V}_1 \coprod \mathbf{V}_2)$ which sends $M \mapsto \gamma_1(M) \uplus \gamma_2(M)$ is injective (and hence bijective).

(b) For (M_1, N_2) an update request from Γ_1 to $\mathbf{D}$, $(M_1, M_2) \in \mathsf{UpdRealiz}\langle M_1, N_2, \Gamma_1\rangle$ is called a *Γ_2-constant realization* of (M_1, N_2) if $\gamma_2(M_1) = \gamma_2(M_2)$.

The following classical observation [1, Sec. 5], which follows immediately from the injectivity of $\gamma_1 \coprod \gamma_2$, is key to the entire strategy.

Observation 3.6. *Let $\{\Gamma_1, \Gamma_2\}$ be a complementary pair, and let (M_1, N_2) be an update request from Γ_1 to $\mathbf{D}$. Then there is at most one Γ_2-constant realization of (M_1, N_2), and this realization exists iff there is an $M_2 \in \mathsf{LDB}(\mathbf{D})$ such that $(\gamma_1 \coprod \gamma_2)(M_2) = N_2 \uplus \gamma_2(M_1)$. In this case, the unique realization is given by (M_1, M_2).* □

4 The Theory of Unique Reflections

In this section, the central results on uniqueness of constant-complement translations are developed.

Notation 4.1. Throughout this section, unless stated specifically to the contrary, take $\mathbf{D}$, $\mathbf{D}_1$, and $\mathbf{D}_2$ to be a relational schema over $(\mathcal{D}, \mathcal{I})$, $\Gamma = (\mathbf{V}, \gamma)$, $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$, and $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ to be views on $\mathbf{D}$, with $\mathcal{C}$ a constraint class.

Definition 4.2 (Information content and separation). A central theme of this work is that the information content of a database may be characterized by the set of sentences from a particular set which it satisfies. Let $\Sigma \subseteq \mathsf{WFS}(\mathbf{D})$ and let $M \in \mathsf{DB}(\mathbf{D})$. The *information content* of M relative to Σ is the set of all sentences in Σ which are true for M. More precisely, $\mathsf{Info}\langle M, \Sigma\rangle = \{\varphi \in \Sigma \mid M \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)\}$. $M_1, M_2 \in \mathsf{DB}(\mathbf{D})$ are *Σ-equivalent* if they have the same information content relative to Σ; i.e., $\mathsf{Info}\langle M_1, \Sigma\rangle = \mathsf{Info}\langle M_2, \Sigma\rangle$. Σ is *separating* for $\mathbf{D}$ if whenever $M_1, M_2 \in \mathsf{LDB}(\mathbf{D})$ are Σ-equivalent, it must be the case that $M_1 = M_2$.

Definition 4.3 (Information-monotone sentences). Intuitively, if $M_1 \subseteq M_2$, then it should be the case that M_2 contains more information than M_1; i.e., $\mathsf{Info}\langle M_1, \Sigma\rangle \subseteq \mathsf{Info}\langle M_2, \Sigma\rangle$. However, in general, M_1 will satisfy constraints which M_2 does not; for example, if $t \in M_2 \setminus M_1$, then $\neg t$ is true of M_1 but not of M_2. If such negative constraints are excluded, then Σ is termed information

monotone. More formally, The sentence $\varphi \in \mathsf{WFS}(\mathbf{D})$ is *information monotone* if for any $M_1, M_2 \in \mathsf{DB}(\mathbf{D})$ if $M_1 \subseteq M_2$, then $\mathsf{Info}\langle M_1, \{\varphi\}\rangle \subseteq \mathsf{Info}\langle M_2, \{\varphi\}\rangle$. In other words, φ is information monotone if $\mathsf{AtMod}_{\mathcal{I}}(\varphi)$ is closed under supersets within $\mathsf{DB}(\mathbf{D})$; whenever $M \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)$, then all $M' \in \mathsf{DB}(\mathbf{D})$ with $M \subseteq M'$ are also in $\mathsf{AtMod}_{\mathcal{I}}(\varphi)$. The set $\Sigma \subseteq \mathsf{WFS}(\mathbf{D})$ is *information monotone* if each $\varphi \in \Sigma$ has this property.

Proposition 4.4 (Key instances of information monotone families). *Any subset of* $\mathsf{WFS}(\mathbf{D}, \exists\neq)$ *is information monotone and separating. This includes, in particular,* $\mathsf{WFS}(\mathbf{D}, \exists+)$, $\mathsf{WFS}(\mathbf{D}, \exists\wedge+)$, *and* $\mathsf{GrAtoms}(\mathbf{D})$.

Proof. It is immediate that any formula involving only existential quantification and not involving any negation is information monotone. That allowing negations of equality atoms does not violate information monotonicity follows from the fact that the equality relation is fixed over all databases. To see this more clearly, consider adding a new, fixed relation $\neq$ which represents inequality explicitly, and replacing each atom of the form $\neg(\tau_1 = \tau_2)$ with $\neq(\tau_1, \tau_2)$. Then, all negation can be removed from the sentences, and so any such subset is information monotone.

To complete the proof, it suffices to observe that any subset of $\mathsf{WFS}(\mathbf{D})$ which contains $\mathsf{GrAtoms}(\mathbf{D})$ is separating, and $\mathsf{GrAtoms}(\mathbf{D})$ is contained in both $\mathsf{WFS}(\mathbf{D}, \exists+)$ and $\mathsf{WFS}(\mathbf{D}, \exists\wedge+)$. □

The proof of the following observation is immediate, but the statement is of such central importance that it is worth noting explicitly.

Observation 4.5 ($\mathsf{Info}\langle M, \Sigma\rangle$ determines M for Σ separating). *Let Σ be an information monotone and separating family on $\mathbf{D}$.*

(a) *For any $M \in \mathsf{LDB}(\mathbf{D})$, M is the least element (under $\subseteq$) of $\mathsf{AtMod}_{\mathcal{I}}(\mathsf{Info}\langle M, \Sigma\rangle) \cap \mathsf{LDB}(\mathbf{D})$.*

(b) *If $M_1, M_2 \in \mathsf{LDB}(\mathbf{D})$, then $M_1 \subseteq M_2$ iff $\mathsf{Info}\langle M_1, \Sigma\rangle \subseteq \mathsf{Info}\langle M_2, \Sigma\rangle$.* □

A central premise of this work is that it is advantageous to view database morphisms as mappings between sentences (in an appropriate information-monotone family), rather than just as mappings from databases to databases. The following definition formalizes this idea.

Definition 4.6 (Substitution of sentences). Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism. The association $t \mapsto \mathsf{Subst}\langle f, t\rangle$ defined in Definition 2.6 may be extended in a natural way to all of $\mathsf{WFS}(\mathbf{D}_2)$. Specifically, the *interpretation of φ in f* is the sentence $\mathsf{Subst}\langle f, \varphi\rangle \in \mathsf{WFS}(\mathbf{D}_1)$ which is obtained by first renaming all quantified variables so that no two formulas involved in the construction have any such variables in common, and then replacing each atom ψ which occurs in φ with $\mathsf{Subst}\langle f, \psi\rangle$. As a specific example, suppose that $\mathbf{D}_1$ has two relation symbols $R_{11}[ABD]$ and $R_{12}[DBC]$, and that $\mathbf{D}_2$ has two relation symbols $R_{21}[AB]$ and $R_{22}[BC]$. Let the defining formulas be $f^{R_{21}} = (\exists x_1)(R_{11}(x_A, x_B, x_1))$ and $f^{R_{22}} = (\exists x_2)(\exists x_3)(R_{11}(x_2, x_B, x_3) \wedge R_{12}(x_3, x_B, x_C))$, with the sentence to be

interpreted $\varphi = (\exists x_4)(R_{21}(a, x_4) \wedge R_{22}(x_4, c))$. Here a and c are constants. Then $\mathsf{Subst}\langle f, \varphi\rangle =$

$$(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4)(R_{11}(a, x_4, x_1) \wedge R_{11}(x_2, x_4, x_3) \wedge R_{12}(x_3, x_4, c)).$$

For more detailed examples, see [10, Sec. 3].

Think of this definition as specifying a function $\mathsf{Subst}\langle f, -\rangle : \mathsf{WFS}(\mathbf{D}_2) \to \mathsf{WFS}(\mathbf{D}_1)$. The key feature to keep in mind is that while the underlying function $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$ maps databases of $\mathbf{D}_1$ to databases of $\mathbf{D}_2$, the interpretation mapping $\mathsf{Subst}\langle f, -\rangle$ sends sentences in $\mathsf{WFS}(\mathbf{D}_2)$ to sentences in $\mathsf{WFS}(\mathbf{D}_1)$. Thus, the direction is reversed. It should perhaps be noted that this definition extends easily to well-formed formulas which are not sentences, but since such an extension is not needed here, it will not be elaborated.

The relationship between the mapping of databases and the mapping of models is a close one, as shown by the following observation, whose straightforward proof is omitted.

Observation 4.7. *Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism, and let $M \in \mathsf{DB}(\mathbf{D}_1)$. Then for any $\varphi \in \mathsf{WFS}(\mathbf{D}_2)$, $f(M) \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)$ iff $M \in \mathsf{AtMod}_{\mathcal{I}}(\mathsf{Subst}\langle f, \varphi\rangle)$.* □

Definition 4.8 (Schemata, morphisms, and views of class $\mathcal{C}$). To measure information content using the sentences of class $\mathcal{C}$, it is important that the database schemata and views respect that class in a certain way.

(a) The database schema $\mathbf{D}$ is *of class* $\mathcal{C}$ if $\mathsf{WFS}(\mathbf{D}, \mathcal{C})$ is separating for $\mathbf{D}$.
(b) The database morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ is *of class* $\mathcal{C}$ if for every $\varphi \in \mathsf{WFF}(\mathbf{D}_2, \mathcal{C})$, $\mathsf{Subst}\langle f, \varphi\rangle \in \mathsf{WFF}(\mathbf{D}_1, \mathcal{C})$.
(c) The view $\Gamma = (\mathbf{V}, \gamma)$ is *of class* $\mathcal{C}$ if both $\mathbf{V}$ and γ are of that class.

In view of Proposition 4.4, any $\mathcal{C}$ which contains $\mathsf{GrAtoms}(\mathbf{D})$ is separating, and hence any database schema $\mathbf{D}$ is of such a class $\mathcal{C}$. However, the notion of a database morphism being of class $\mathcal{C}$ is more complex, and warrants closer attention via a few examples.

Example 4.9 (Morphisms of class $\mathcal{C}$). Let $\mathbf{E}_1$ be the relational schema with two ternary relation symbol $R_{11}[ABC]$ and $R_{12}[ABC]$, and let $\mathbf{E}_2$ be the schema with the single relation symbol $R_2[AB]$. Define the morphism $g_{11} : \mathbf{E}_1 \to \mathbf{E}_2$ by $g_{11}^{R_2} = (\exists z)(R_{11}(x_A, x_B, z) \wedge R_{12}(x_A, x_B, z))$. In other words, R_2 is the projection of the intersection of R_{11} and R_{12}. Then g_{11} is of class $\mathcal{C}$ for $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathbf{1}\}$, but not for $\mathcal{C} = \mathsf{Atoms}$, since $\mathsf{Subst}\langle g_{11}^{R_2}, t\rangle$ is not equivalent to a ground atom for $t \in \mathsf{GrAtoms}(\mathbf{E}_2)$. On the other hand, define $g_{12} : \mathbf{E}_1 \to \mathbf{E}_2$ by $g_{12}^{R_2} = (\exists z)(R_{11}(x_A, x_B, z) \wedge (\neg R_{12}(x_A, x_B, z)))$. In this case, R_2 is the projection of the difference of R_{11} and R_{12}, and g_{12} is of class $\mathcal{C}$ for $\mathcal{C} = \mathbf{1}$, but not for any of the others listed above, since the sentence $\mathsf{Subst}\langle g_{12}^{R_2}, \varphi\rangle$ will always contain non-removable internal negation, even for φ a ground atom. In particular, g_{12} is not of class $\exists\wedge+$. Finally, define $g_{13} : \mathbf{E}_1 \to \mathbf{E}_2$ by $g_{11}^{R_2} = (\exists z)(R_{11}(x_A, x_B, z) \vee R_{12}(x_A, x_B, z))$. In other words, R_2 is the projection of the union of R_{11} and R_{12}. Then g_{13} is of class $\mathcal{C}$ for $\mathcal{C} \in \{\exists\neq, \exists+, \mathbf{1}\}$, but not for $\mathcal{C} \in \{\exists\wedge+, \mathsf{Atoms}\}$, since $\mathsf{Subst}\langle g_{13}, \varphi\rangle$ will always contain non-removable disjunction, even for φ a ground atom.

Notation 4.10. For the remainder of this section, unless stated specifically to the contrary, take all database schemata to be of class $\mathcal{C}$.

Definition 4.11 (Semantic morphisms). Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism of class $\mathcal{C}$. In addition to the standard notions of injectivity, surjectivity, and bijectivity for the induced mapping $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$, there are corresponding notions associated with the substitution mapping $\mathsf{Subst}\langle f, - \rangle : \mathsf{WFS}(\mathbf{D}_2, \mathcal{C}) \to \mathsf{WFS}(\mathbf{D}_1, \mathcal{C})$. The point of care to be taken is that the identification is only unique up to the equivalence $\equiv_{\mathbf{D}}$ as defined in Definition 2.5.

(a) f is *semantically injective for* $\mathcal{C}$ if for every $\varphi_1, \varphi_2 \in \mathsf{WFS}(\mathbf{D}_2, \mathcal{C})$, if $\mathsf{Subst}\langle f, \varphi_1 \rangle \equiv_{\mathbf{D}_1} \mathsf{Subst}\langle f, \varphi_2 \rangle$, then $\varphi_1 \equiv_{\mathbf{D}_2} \varphi_2$.

(b) f is *semantically surjective for* $\mathcal{C}$ if for every $\varphi_1 \in \mathsf{WFS}(\mathbf{D}_1, \mathcal{C})$, there is a $\varphi_2 \in \mathsf{WFS}(\mathbf{D}_2, \mathcal{C})$ with $\mathsf{Subst}\langle f, \varphi_2 \rangle \equiv_{\mathbf{D}_1} \varphi_1$.

(c) f is *semantically bijective for* $\mathcal{C}$ if it is both semantically injective and semantically surjective for $\mathcal{C}$.

A view $\Gamma = (\mathbf{V}, \gamma)$ is said to be *semantically bijective for* $\mathcal{C}$ precisely when the morphism γ has that property.

It should be stressed that for f to be of class $\mathcal{C}$ is *causa sine qua non* to have any of the above properties. If f is not of class $\mathcal{C}$, then it is not semantically injective, surjective, or bijective for $\mathcal{C}$, by definition.

It is easy to see that the morphisms g_{11}, g_{12}, and g_{13} of Example 4.9 are semantically injective; that is, logically distinct formulas in $\mathbf{E}_2$ give rise to logically distinct formulas in $\mathbf{E}_1$. This is true more generally; surjectivity of the underlying database mapping translates to semantic injectivity *provided the morphism is of the appropriate class.*

Proposition 4.12 (Underlying surjectivity $\Rightarrow$ semantic injectivity). *Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism of class $\mathcal{C}$. If the associated function $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$ is surjective, then f is semantically injective for $\mathcal{C}$. In particular, for every view $\Gamma = (\mathbf{V}, \gamma)$ of class $\mathcal{C}$, the morphism γ is semantically injective for $\mathcal{C}$.*

Proof. Let $\varphi_1, \varphi_2 \in \mathsf{WFS}(\mathbf{D}_2, \mathcal{C})$ with $\varphi_1 \not\equiv_{\mathbf{D}_2} \varphi_2$. Then there exists an $N \in \mathsf{LDB}(\mathbf{D}_2)$ which is an atomic model of one but not the other. Without loss of generality, assume that $N \in \mathsf{AtMod}_{\mathcal{I}}(\varphi_1) \setminus \mathsf{AtMod}_{\mathcal{I}}(\varphi_2)$. Then for any $M \in f^{-1}(N)$, $M \in \mathsf{AtMod}_{\mathcal{I}}(\mathsf{Subst}\langle f, \varphi_1 \rangle) \setminus \mathsf{AtMod}_{\mathcal{I}}(\mathsf{Subst}\langle f, \varphi_2 \rangle)$. Hence, $\mathsf{Subst}\langle f, \varphi_1 \rangle \not\equiv_{\mathbf{D}_1} \mathsf{Subst}\langle f, \varphi_2 \rangle$, and so f is semantically injective. □

For this work, semantic surjectivity on its own is not of central importance. Rather, the key property is semantic bijectivity. Examples illustrating these ideas are found in Example 4.16 below. First, however, it is essential to introduce some supporting ideas.

The semantic bijectivity of a morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ entails nothing more than a bijective correspondence between the equivalence classes of sentences of class $\mathcal{C}$ in $\mathbf{D}_1$ and those of $\mathbf{D}_2$. This is formulated precisely as follows.

Observation 4.13 (Characterization of semantic bijectivity). *Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be database morphism of class $\mathcal{C}$. Then f if semantically bijective for $\mathcal{C}$*

iff it induces a natural bijection between $\mathsf{WFF}(\mathbf{D}_2, \mathcal{C})/\equiv_{\mathbf{D}_2}$ *and* $\mathsf{WFF}(\mathbf{D}_1, \mathcal{C})/\equiv_{\mathbf{D}_1}$ *via* $[\varphi]_{\equiv_{\mathbf{D}_2}} \mapsto [\mathsf{Subst}\langle f, \varphi\rangle]_{\equiv_{\mathbf{D}_1}}$. □

Definition 4.14 (The reconstruction morphism). Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism. If the induced function $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$ is bijective, then there is trivially a function $g : \mathsf{LDB}(\mathbf{D}_2) \to \mathsf{LDB}(\mathbf{D}_1)$ for which is inverse to f. What is remarkable is that there is database morphism $g : \mathbf{D}_2 \to \mathbf{D}_1$ in the logical sense which induces g. This is a consequence of Beth's theorem from model theory [12, Thm. 22.4]. This g is called the *reconstruction morphism* for f and is denoted $\mathsf{RcMor}(f)$.

Observe that f and $\mathsf{RcMor}(f)$ are inverses for the mappings on sentences as well, up to the logical equivalence defined by the schemata. More precisely, for any $\varphi_1 \in \mathsf{WFS}(\mathbf{D}_1)$, $\mathsf{Subst}\langle f, \mathsf{Subst}\langle \mathsf{RcMor}(f), \varphi_1\rangle\rangle \equiv_{\mathbf{D}_1} \varphi_1$, and for any $\varphi_2 \in \mathsf{WFS}(\mathbf{D}_2)$, $\mathsf{Subst}\langle \mathsf{RcMor}(f), \mathsf{Subst}\langle f, \varphi_2\rangle\rangle \equiv_{\mathbf{D}_1} \varphi_2$.

Unfortunately, there is no guarantee that $\mathsf{RcMor}(f)$ will be of the same class as f. When it is, however, semantic bijectivity is guaranteed.

Proposition 4.15 (Semantic bijectivity ⇔ class $\mathcal{C}$ reconstruction). *Let* $f : \mathbf{D}_1 \to \mathbf{D}_2$ *be a database morphism of class* $\mathcal{C}$ *whose underlying function* $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$ *is bijective. Then* f *is semantically bijective for* $\mathcal{C}$ *iff* $\mathsf{RcMor}(f)$ *is of class* $\mathcal{C}$.

Proof. Let $\varphi_1 \in \mathsf{WFS}(\mathbf{D}_1, \mathcal{C})$. If $\mathsf{RcMor}(f)$ is of class $\mathcal{C}$, then there is a $\varphi_2 \in \mathsf{WFS}(\mathbf{D}_2, \mathcal{C})$ such that $\mathsf{Subst}\langle \mathsf{RcMor}(f), \varphi_1\rangle \equiv_{\mathbf{D}_2} \varphi_2$, and since $\mathsf{Subst}\langle f, -\rangle \circ \mathsf{Subst}\langle \mathsf{RcMor}(f), -\rangle$ is the identity on $\mathsf{WFS}(\mathbf{D}_1)$, up to equivalence of $\equiv_{\mathbf{D}_1}$, $\mathsf{Subst}\langle f, \varphi_2\rangle \equiv_{\mathbf{D}_1} \varphi_1$. Hence f is semantically surjective, and since it is semantically injective by Proposition 4.12, it is semantically bijective for $\mathcal{C}$.

Conversely, if f is semantically bijective for $\mathcal{C}$, then given $\varphi_1 \in \mathsf{WFS}(\mathbf{D}_1, \mathcal{C})$, there is a $\varphi_2 \in \mathsf{WFS}(\mathbf{D}_2, \mathcal{C})$ such that $\mathsf{Subst}\langle f, \varphi_2\rangle \equiv_{\mathbf{D}_1} \varphi_1$. Since $\mathsf{Subst}\langle \mathsf{RcMor}(f), -\rangle \circ \mathsf{Subst}\langle f, -\rangle$ is the identity on $\mathsf{WFS}(\mathbf{D}_2)$, up to equivalence of $\equiv_{\mathbf{D}_2}$, it must be the case that $\mathsf{Subst}\langle \mathsf{RcMor}(f), \varphi_1\rangle \equiv_{\mathbf{D}_2} \varphi_2$, whence $\mathsf{RcMor}(f)$ is of class $\mathcal{C}$. □

Example 4.16 (Semantic bijectivity and reconstruction morphisms). It is clear from Proposition 4.15 that every bijective morphism is semantically bijective for class **1**; that is, when all first-order sentences are considered. However, this is not a very useful property within the context of this work, since the set of all first-order sentences on a schema is not information monotone except in trivial cases. In particular, owing to a special property to be developed in Proposition 4.18, the choice $\mathcal{C} = \exists\wedge+$ will yield the most fruitful results. It is therefore necessary to establish useful conditions under which semantic bijectivity holds for more restricted classes. To set the stage for this, some illustrative examples based upon a set of four schemata are presented. Let $\mathbf{E}_3$ be the relational schema with three unary relation symbols $R_{31}[A]$, $R_{32}[A]$, and $R_{33}[A]$, constrained by the single sentence $(\forall x)(R_{33}(x) \Leftrightarrow R_{31}(x) \wedge R_{32}(x))$. In other words, the state of R_{33} is the intersection of that of R_{31} and R_{32}. Let $\mathbf{E}_4$ have a corresponding set of three unary relation symbols $R_{41}[A]$, $R_{42}[A]$, and $R_{43}[A]$, but this time constrained

by the single sentence $(\forall x)(R_{43}(x) \Leftrightarrow ((R_{41}(x) \wedge \neg R_{42}(x)) \vee (\neg R_{42}(x) \wedge R_{41}(x))))$. In other words, the state of R_{43} is the symmetric difference of that of R_{41} and R_{42}. Define $\mathbf{E}_5$ to have the two unary symbols $R_{51}[A]$ and $R_{52}[A]$, with no other constraints, and define $\mathbf{E}_6$ to have the two unary relation symbols $R_{61}[A]$ and $R_{63}[A]$, again with no other constraints.

First of all, consider the morphism $h_3 : \mathbf{E}_3 \to \mathbf{E}_5$ which identifies R_{51} with R_{31} and R_{52} with R_{32}. Formally, $h_3^{R_{51}} = R_{31}(x_A)$ and $h_3^{R_{52}} = R_{32}(x_A)$. It is trivial that h_3 is of class $\mathcal{C}$ for any $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathsf{Atoms}, \mathbf{1}\}$. The reconstruction mapping for h_3 is $g_3 : \mathbf{E}_5 \to \mathbf{E}_3$ defined by $g_3^{R_{31}} = R_{51}(x_A)$, $g_3^{R_{32}} = R_{52}(x_A)$, and $g_3^{R_{33}} = R_{51}(x_A) \wedge R_{52}(x_A)$. This morphism is of class $\mathcal{C}$ for $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathbf{1}\}$, but not for $\mathcal{C} = \mathsf{Atoms}$, since the interpretation formula for R_{33} is not equivalent to any atomic formula. Consequently, h_3 and g_3 are semantic bijections for $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathbf{1}\}$.

Next, consider the morphism $h_4 : \mathbf{E}_4 \to \mathbf{E}_5$ which identifies R_{51} with R_{41} and R_{52} with R_{42}. Formally, $h_4^{R_{51}} = R_{31}(x_a)$ and $h_4^{R_{52}} = R_{32}(x_a)$. Just as was the case for h_3, this morphism h_4 is is of class $\mathcal{C}$ for any $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathsf{Atoms}, \mathbf{1}\}$. The reconstruction mapping is $g_4 : \mathbf{E}_5 \to \mathbf{E}_4$ given by $g_4^{R_{41}} = R_{51}(x_A)$, $h_4^{R_{42}} = R_{52}(x_A)$, and $g_4^{R_{43}} = (R_{51}(x_A) \wedge (\neg R_{52}(x_A))) \vee (R_{52}(x_A) \wedge (\neg R_{51}(x_A)))$. This time, amongst the possibilities $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathsf{Atoms}, \mathbf{1}\}$, the reconstruction mapping g_4 is of class $\mathcal{C}$ only for $\mathcal{C} = \mathbf{1}$. The definition of $g_4^{R_{43}}$ excludes all other possibilities.

A similar failure of the reconstruction morphism to be of classes other than $\mathbf{1}$ may occur even when the schemata themselves have no nontrivial constraints. For example, let $h_5 : \mathbf{E}_5 \to \mathbf{E}_6$ be defined by $h_5^{R_{61}} = R_{51}(x_A)$ and $h_5^{R_{63}} = (R_{51}(x_A) \wedge (\neg R_{52}(x_A))) \vee (R_{51}(x_A) \wedge (\neg R_{51}(x_A)))$. In other words, R_{63} constrained by the interpretation morphism to be the symmetric difference of R_{51} and R_{52}. It is easily seen that h_5 is bijective on databases, with the reconstruction morphism $g_5 : \mathbf{E}_6 \to \mathbf{E}_5$ defined by $g_5^{R_{51}} = R_{61}(x_A)$ and $g_5^{R_{52}} = (R_{61}(x_A) \wedge (\neg R_{63}(x_A))) \vee (R_{63}(x_A) \wedge (\neg R_{61}(x_A)))$. However, amongst the possibilities $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathsf{Atoms}, \mathbf{1}\}$, both h_5 and g_5 are of class $\mathcal{C}$ only for $\mathcal{C} = \mathbf{1}$, and so cannot possibly be semantically bijective for any other of the classes.

These examples suggest that for a morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ which is bijective on databases to fail to be semantically bijective, there must be some nonmonotonicity of information, either in the constraints of the domain schema $\mathbf{D}_1$, or else in the interpretation formulas defined by the interpretation morphism itself. That this is indeed the case, at least for certain common contexts, will now be shown.

Definition 4.17 (Universal models). Traditional database dependencies take the form of generalized universal-existential Horn clauses of the following form.

$$(\forall x_1)(\forall x_2)\ldots(\forall x_n)((A_1 \wedge A_2 \wedge \ldots \wedge A_n) \Rightarrow (\exists y_1)(\exists y_2)\ldots(\exists y_r)(B_1 \wedge B_2 \wedge \ldots \wedge B_s))$$

The A_i's and the B_i's are atoms, but of course not ground atoms. indeed, the quantified variables must occur in these atoms. There is a rich variety of alternatives; for a detailed taxonomy and comparison of properties consult [13].

Given a set S of ground atoms, one may attempt to construct a least model containing S by using so-called forward chaining — repeatedly unifying the left-hand side (the A_i's) with known facts (qua ground atoms) in S in order to deduce new ones from the right-hand side (the B_i's). The new facts are added to S and the process is repeated until no new rules apply. This is a classical form of inference for propositional Horn clauses [14]. It also applies to so-called universal models the first-order case, but with some limitations. In its general form, it has seen recent application in the context of data exchange [15] and in the realization of canonical reflections for view updates which lie outside of the scope of constant complement [5]. The process is, in turn, based upon the classical chase inference procedure [16].

There are a few complications relative to the propositional setting. First of all, it may be necessary to generate "generic" constants, because of the existential quantifiers, so the least model may only be unique up to a suitable renaming of such constants. Second, the process may not always terminate, but rather continue endlessly to generate new tuples [15, Example 3.6] [5, 4.14]. Nevertheless, there are wide classes of constraints for which the procedure is known to terminate. One possibility is to work with *full dependencies* which do not involve any existential quantification. A broader solution is to work with the so-called *weakly acyclic* tuple-generating dependencies (tgds), together with the classical equality-generating dependencies (egds) [15, Thm. 3.9].

When they exist, such universal models have a simple characterization [5, Sec. 3] [15, Sec. 3.1]. An *endomorphism* on $\mathcal{D}$ is a function $h : \mathsf{Const}(\mathcal{D}) \to \mathsf{Const}(\mathcal{D})$ which preserves attribute types, in the precise sense that for each $A \in \mathcal{A}_\mathcal{D}$ and each $a \in \mathsf{Const}_\mathcal{D}(A)$, $h(a) \in \mathsf{Const}_\mathcal{D}(A)$. If h is additionally a bijection, then it is called an *automorphism* of $\mathcal{D}$. For $S \subseteq \mathsf{Const}(\mathcal{D})$, call h S-invariant if $h(a) = a$ for all $a \in S$. Given a database schema $\mathbf{D}$, an endomorphism on $\mathcal{D}$ induces a mapping from $\mathsf{GrAtoms}(\mathbf{D})$ to itself given by sending $t \in \mathsf{GrAtoms}(\mathbf{D})$ to the tuple t' with $t'[\mathsf{RName}] = t[\mathsf{RName}]$ and $t'[A] = t[h(A)]$ for all $A \in \mathsf{Ar}_{\mathbf{t}[\mathsf{RName}]}$. This mapping on atoms is also represented by h, as will the induced mapping from $\mathsf{DB}(\mathbf{D})$ to itself given by $M \mapsto \{h(t) \mid t \in M\}$. $h(M)$ is called an *endomorphic image* of M. Given $\Phi \subseteq \mathsf{WFS}(\mathbf{D})$, an $M \in \mathsf{DB}(\mathbf{D})$ is a *universal model* for Ψ if every $M \in \mathsf{AtMod}_\mathcal{I}(\Psi)$ is a superset of an endomorphic image of M.

Say that $\mathbf{D}$ *admits universal models* if $M \cup \mathsf{Constr}(\mathbf{D})$ admits a universal model for every $M \in \mathsf{DB}(\mathbf{D})$ which extends to a model; i.e., for which there exists an $M' \in \mathsf{LDB}(\mathbf{D})$ with $M \subseteq M'$.

In the context of database constraints, such universal models may be generated using the inference procedure described above [15, Sec. 3.1], provided the procedure terminates. In particular, the combination of weakly acyclic tgds and all egds noted above has this property. The following result thus provides a rich class of base schemata which imply semantic bijectivity for $\exists\wedge+$ when the underlying function is bijective.

Proposition 4.18 (Universal models ⇒ semantic bijectivity). *Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ be a database morphism of class $\exists\wedge+$ whose underlying function*

$f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$ *is bijective. If* $\mathbf{D}_1$ *admits universal models, then* f *is semantically bijective for* $\exists\wedge+$.

PROOF OUTLINE: Space limitations preclude a detailed proof, but it is easy to sketch the main idea. Let $\varphi_1 \in \mathsf{WFS}(\mathbf{D}_1, \exists\wedge+)$. The basic strategy is to Skolemize φ_1 into a set G of ground atoms by replacing each existentially-quantified variable by a distinct new constant not appearing in $\mathsf{Constr}(\mathbf{D})$, and to generate a universal model $M \in \mathsf{LDB}(\mathbf{D}_1)$ for G. Next, map M to $f(M) \in \mathsf{LDB}(\mathbf{D}_2)$ and reverse the process. Represent $f(M)$ as a sentence φ_2' which is the conjunction of the atoms in $f(M)$, and then "un-Skolemize" φ_2' by replacing all constants which were not in the original φ_1 or in $\mathsf{Constr}(\mathbf{D})$ by existentially quantified variables. Call the resulting formula φ_2. It is not difficult to see that $\mathsf{Subst}\langle f, \varphi_2\rangle \equiv_{\mathbf{D}_1} \varphi_1$, from which the result follows. □

In the context of Example 4.16 above, note that for $\mathbf{E}_4$ the constraint $(\forall x)(T_4(x) \Leftrightarrow ((R_4(x)\wedge\neg S_4(x))\vee(\neg S_1(x)\wedge R_4(x))))$ is not a dependency of the $(\forall)(\exists)$-Horn variety, and does not admit universal models. On the other hand, the alternative $(\forall x)(T_3(x) \Leftrightarrow R_3(x)\wedge S_3(x))$ for $\mathbf{E}_3$ is in fact representable as as set of three tgds: $(\forall x)(T_3(x) \Rightarrow R_3(x))$, $(\forall x)(T_3(x) \Rightarrow S_3(x))$, and $(\forall x)((R_3(x)\wedge S_3(x)) \Rightarrow T_3(x))$, and so does admit universal models by [15, Thm. 3.9]. Thus, the assertion of the above proposition is confirmed by this example.

Definition 4.19 (Update difference and optimal reflections). The update difference of an update (M_1, M_2) on $\mathbf{D}$ with respect to a set $\Sigma \subseteq \mathsf{WFS}(\mathbf{D})$ is a measure of how much M_1 and M_2 differ in terms of their information content relative to Σ. Formally, the *positive* (Δ^+)*, negative* (Δ^-)*, and total* (Δ) *update differences* of (M_1, M_2) with respect to Σ are defined as follows:

$$\begin{aligned}\Delta^+\langle(M_1, M_2), \Sigma\rangle &= \mathsf{Info}\langle M_2, \Sigma\rangle \setminus \mathsf{Info}\langle M_1, \Sigma\rangle\\ \Delta^-\langle(M_1, M_2), \Sigma\rangle &= \mathsf{Info}\langle M_1, \Sigma\rangle \setminus \mathsf{Info}\langle M_2, \Sigma\rangle\\ \Delta\langle(M_1, M_2), \Sigma\rangle &= \Delta^+\langle(M_1, M_2), \Sigma\rangle \cup \Delta^-\langle(M_1, M_2), \Sigma\rangle\end{aligned}$$

Given $\varphi \in \Delta\langle(M_1, M_2), \Sigma\rangle$, it is always possible to determine whether $\varphi \in \Delta^+\langle(M_1, M_2), \Sigma\rangle$ or $\varphi \in \Delta^-\langle(M_1, M_2), \Sigma\rangle$ by checking whether or not $M_1 \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)$.

For an update request (M_1, N_2) from Γ to $\mathbf{D}$, the quality of a realization (M_1, M_2) is measured by its update difference, with an optimal realization one which entails the least change of information. Formally, let $\Sigma \subseteq \mathsf{WFS}(\mathbf{D})$, let (M_1, N_2) be an update request along Γ, and let $(M_1, M_2) \in \mathsf{UpdRealiz}\langle M_1, N_2, \Gamma\rangle$. The pair (M_1, M_2) is *optimal* with respect to Σ if for all $(M_1, M_2') \in \mathsf{UpdRealiz}\langle M_1, N_2, \Gamma\rangle$, $\Delta\langle(M_1, M_2), \Sigma\rangle \subseteq \Delta\langle(M_1, M_2'), \Sigma\rangle$.

The definition of optimal which is used here is slightly different than that of [5, 3.5], in which optimality also requires minimality of update difference with respect to $\mathsf{GrAtoms}(\mathbf{D})$. That additional condition was necessary because the main information measure was not required to be separating. Here, the separating condition effectively provides the additional minimality.

Lemma 4.20 (Distance preservation under semantic bijection). *Let* $f : \mathbf{D}_1 \to \mathbf{D}_2$ *be a semantic bijection of class* $\mathcal{C}$*, and* $M_1, M_2, M_3 \in \mathsf{LDB}(\mathbf{D}_1)$*. Then*

$$\Delta\langle(M_1, M_2), \mathsf{WFF}(\mathbf{D}_1, \mathcal{C})\rangle \subseteq \Delta\langle(M_1, M_3), \mathsf{WFF}(\mathbf{D}_1, \mathcal{C})\rangle$$
iff $\quad \Delta\langle(f(M_1), f(M_2)), \mathsf{WFF}(\mathbf{D}_2, \mathcal{C})\rangle \subseteq \Delta\langle(f(M_1), f(M_3)), \mathsf{WFF}(\mathbf{D}_2, \mathcal{C})\rangle.$

Proof. The proof follows directly from the bijective correspondence between (semantic equivalence classes of) sentences in $\mathsf{WFF}(\mathbf{D}_1, \mathcal{C})$ and those in $\mathsf{WFF}(\mathbf{D}_2, \mathcal{C})$, as established in Observation 4.13. □

Lemma 4.21. *If the views* $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ *are of class* $\mathcal{C}$*, then so too is* $\Gamma_1 \coprod \Gamma_2$*.*

Proof. This follows from the definition (Definition 4.8(b)), since the set of interpretation functions for $\mathbf{V}_1 \coprod \mathbf{V}_2$ into $\mathbf{D}$ is simply the (disjoint) union of those for γ_1 and for γ_2. □

The definition of complementary views Definition 3.5 is extended to the $\mathcal{C}$ context in the following fashion.

Definition 4.22 ($\mathcal{C}$-complementary pairs). The set $\{\Gamma_1, \Gamma_2\}$ of views is said to be a *$\mathcal{C}$-complementary pair* if each view is of class $\mathcal{C}$ and, in addition, the morphism $\gamma_1 \coprod \gamma_2 : \mathbf{D} \to \mathbf{V}_1 \coprod \mathbf{V}_2$ is semantically bijective for $\mathcal{C}$.

Finally, it is possible to establish the main result of this paper.

Theorem 4.23. *Let* $\{\Gamma_1 = (\mathbf{V}_1, \gamma_1), \Gamma_2 = (\mathbf{V}_2, \gamma_2)\}$ *be a* $\mathcal{C}$*-complementary pair and let* (M_1, N_1) *be an update request from* Γ_1 *to* $\mathbf{D}$*. If the* Γ_2*-constant realization of* (M_1, N_1) *along* Γ_1 *exists, it is optimal with respect to* $\mathsf{WFS}(\mathbf{D}, \mathcal{C})$*.*

Proof. Let $M_2 \in \mathsf{LDB}(\mathbf{D})$ be the unique database for which (M_1, M_2) is the Γ_2 constant realization of (M_1, N_1); thus, $(\gamma_1 \coprod \gamma_2)(M_2) = (N_1, \gamma_2(M_1))$. Let $M_3 \in \mathsf{LDB}(\mathbf{D})$ be any database for which $\gamma_1(M_3) = N_1'$; thus, (M_1, M_3) is an arbitrary realization of the update request (M_1, N_1'). The update in $\mathbf{V}_1 \coprod \mathbf{V}_2$ corresponding to (M_1, M_2) under $\gamma_1 \coprod \gamma_2$ is $u = ((\gamma_1(M_1) \uplus \gamma_2(M_1)), (N_1 \uplus \gamma_2(M_1)))$, while for (M_1, M_3) it is $u' = ((\gamma_1(M_1) \uplus \gamma_2(M_1)), (N_1 \uplus \gamma_2(M_3)))$. Clearly $\Delta\langle u, \mathsf{WFS}(\mathbf{V}_1 \coprod \mathbf{V}_2, \mathcal{C})\rangle \subseteq \Delta\langle u', \mathsf{WFS}(\mathbf{V}_1 \coprod \mathbf{V}_2, \mathcal{C})\rangle$, and so in view of Lemma 4.20 and Lemma 4.21, $\Delta\langle(M_1, M_2), \mathsf{WFF}(\mathbf{D}, \mathcal{C})\rangle \subseteq \Delta\langle(M_1, M_3), \mathsf{WFF}(\mathbf{D}, \mathcal{C})\rangle$ as well. Hence (M_1, M_2) is optimal, as required. □

Corollary 4.24 (Global uniqueness of constant-complement updates). *Let* $\{\Gamma_1, \Gamma_2\}$ *and* $\{\Gamma_1, \Gamma_2'\}$ *be* $\mathcal{C}$*-complementary pairs, and let* (M_1, N_2) *be an update request from* Γ_1 *to* $\mathbf{D}$*. If* (M_1, N_2) *has both a* Γ_2*-constant realization and a* Γ_2'*-constant realization, then these two realizations are identical. In other words, a constant-complement realization of an update is independent of the choice of complement, as long as the complementary pair is* $\mathcal{C}$*-compatible.*

Proof. By construction, an optimal reflection is unique, and so the result follows from Theorem 4.23. □

Corollary 4.25 (Universal solutions imply global uniqueness for $\mathcal{C} = \exists\wedge+$). *Assume that* $\mathbf{D}$ *admits universal solutions, let* Γ_1 *be any view of* $\mathbf{D}$ *which is of class* $\exists\wedge+$*, and let* (M_1, N_2) *be an update request from* Γ_1 *to* $\mathbf{D}$*. Then for all views* Γ_2 *of class* $\exists\wedge+$ *which are complements of* Γ_1 *and for which the* Γ_2*-constant translation of* (M_1, N_1) *exists, these translations are identical.*

Proof. The proof follows directly from Proposition 4.18 andCorollary 4.24. □

Example 4.26. To begin, consider an example which was introduced in [17, 1.1.1] as motivation for the need to consider restrictions on the nature of "good" complements. It recaptures ideas similar to those found in $\mathbf{E}_4$ of Example 4.16, but in the context of three views, each of which contains one unary relation symbol. Let $\mathbf{E}_7$ have two relation symbols $R_{71}[A]$ and $R_{72}[A]$, with no constraints other than those imposed by the relational context $\mathcal{D}$. Let $\Omega_{7i} = (\mathbf{W}_{7i}, \omega_{7i})$ for $i \in \{1, 2\}$ be the view which retains R_{7i} but discards $R_{7(3-i)}$. $R_{7i}[A]$ is thus the sole relation symbol for $\mathbf{W}_{7i}$. In each case, the interpretation formula $\omega_{7i}^{R_{7i}}$ is the identity on R_{7i}, and so the coproduct morphism $\omega_{71} \coprod \omega_{72}$ is also an identity and trivially semantically bijective for any reasonable choice for $\mathcal{C}$. The pair $\{\Omega_{71}, \Omega_{72}\}$ is as well behaved as can be, and $\mathcal{C}$-complementary. Now consider a third view $\Omega_{73} = (\mathbf{W}_{73}, \omega_{73})$ which has the single relation symbol $R_{73}[A]$, defined by the formula $\omega_{73}^{R_{73}} = (R_{71}(x_A) \wedge \neg R_{72}(x_A) \vee (\neg R_{71}(x_A) \wedge R_{72}(x_A))$. In other words, the value for R_3 is the symmetric difference of those for R_{71} and R_{72}. It is easy to see that any set of two of these three views forms a complementary pair, but the two pairs which contain Ω_{73} are not $\mathcal{C}$-complementary for $\mathcal{C} \in \{\exists\neq, \exists+, \exists\wedge+, \mathsf{Atoms}\}$. The interpretation $\omega_{73}^{R_3}$ involves negation and so Ω_{73} can never be of class $\mathcal{C}$ for any $\mathcal{C}$ which renders $\mathsf{WFS}(\mathbf{E}_7, \mathcal{C})$ information monotone. Thus, this "undesirable" complement is excluded by the theory. In this example, the schema $\mathbf{E}_7$ admits universal solutions, but the morphism $\omega_{7i} \coprod \omega_{23}$ is not of class $\mathcal{C}$ for $i \in \{1, 2\}$, and so Corollary 4.25 is not applicable.

As a slight variation, reconsider the schema $\mathbf{E}_4$ of Example 4.16, this time with the three views $\Omega_{4i} = (\mathbf{W}_{4i}, \omega_{4i})$, with Ω_{4i} for $i \in \{1, 2, 3\}$ the view which retains R_{4i} but discards the other two relations. Each view morphism ω_{4i} is very well behaved; each is semantically surjective for any choice of $\mathcal{C}$ listed in Definition 2.4. Furthermore, it is easy to see that any two of these views forms a complementary pair. However, by an argument virtually identical to that already given in Example 4.16, for any pair of these views, the reconstruction morphism cannot be of class $\mathcal{C}$, since including R_{43} in the view forces a symmetric difference interpretation. In this case, the each view morphism of the form $\omega_{4i} \coprod \omega_{4j}$ is of class $\mathcal{C}$, but the constraints of the main schema $\mathbf{E}_4$ do not allow the reconstruction to be of class $\mathcal{C}$ for any reasonable choice.

Example 4.27 (Comparison to the order-view approach). Upon comparing Corollary 4.24 of this paper to [3, 4.3], two key differences are apparent. On the one hand, the theory of [3] requires *order complements*, which is based upon the order structure on the states of the schemata and has nothing whatever to do with logic or the relational model, while the theory presented here requires $\mathcal{C}$-complements, which are logic based. Although a detailed study has

not been made, partly because the theory of [3] is limited to so-called meet complements, it does appear the notion of order complement is strictly more general than that of a $\mathcal{C}$-complement. On the other hand, the uniqueness theory of [3] is limited to *order-realizable updates*; that is, updates which can be realized as sequences of legal insertions and deletions. The theory of this paper imposes no such limitation.

To illustrate this advantage, consider the schema $\mathbf{E}_8$ which has the single relation symbol $R[ABC]$, constrained by the functional dependencies $B \to C$ and $B \to A$. The two views are $\Pi^{\mathbf{E}_8}_{AB}$ and $\Pi^{\mathbf{E}_8}_{BC}$, the projections onto AB and BC respectively. Both of these are *order views*; the associated mappings on database states are open poset morphisms. Thus, the result [3, 4.3] applies, but only to order-realizable updates, of which there are none for $\Pi^{\mathbf{E}_8}_{AB}$. More precisely, the only possible updates on $\Pi^{\mathbf{E}_8}_{AB}$ are those which change the A-value of a given tuple, and none of those is an order-realizable update, so that theory does not address this situation in a systematic way [3, 4.6]. On the other hand, the theory of this paper imposes no such restriction to order-realizable updates. Since the coproduct $\Pi^{\mathbf{E}_8}_{AB} \coprod \Pi^{\mathbf{E}_8}_{BC}$ is easily seen to be semantically bijective for any reasonable choice for $\mathcal{C}$ (the reconstruction map is the join), all $\Pi^{\mathbf{E}_8}_{BC}$-constant updates on $\Pi^{\mathbf{E}_8}_{AB}$ are allowed.

5 Conclusions and Further Directions

The constant-complement update strategy for views has been examined from the point of view of information content. Specifically, in this approach, the decomposition mapping for the pair of views is required not only to be bijective on states but also on the sentences which define the information content of the component views. From this perspective, it has been shown that under very broad conditions, the translation of the view update is independent of the choice of complement. In particular, in a traditional database context — well-behaved dependencies and view mappings defined by conjunctive queries — the translation never depends upon the choice of complement.

Further investigations are appropriate for the following topics.

Extension to other logical models The theory presented here is couched squarely within the classical relational model. In principle, it applies as well to other models which admit a first-order logical formalism, such as the nested relational model [6, Ch. 7] and the Higher-Order Entity-Relationship Model (HERM) [18]. However, the extent to which the cornerstone ideas such as identifying a suitable class $\mathcal{C}$ or characterizing semantic bijectivity via universal models translate to realistic data modelling within those frameworks requires further investigation.

Rapprochement with the order-based approach The order-based framework for the constant-complement approach, as reported in [3], has the great advantage that it is not tied to a particular data model. Rather, schemata are modelled as ordered sets and view mappings as poset morphisms. While the framework

presented here is much more specific, being limited to the relational model, it also supports a broader result which is not limited to order-realizable updates. Each approach has its advantages and disadvantages. A rapprochement of the two appears to be a fruitful topic for future investigation. In particular, it would be interesting to identify the extent to which the ideas of this paper can be recast without the specific need for an underlying logic. In this regard, element-based variations of the order-based model, such as that employed in [19], would perhaps form the foundation for a fruitful common ground.

Information morphisms and the inversion of schema mappings Recently, there have been significant advances in the theory of data exchange and inversion of schema mappings [15] [20]. Although these topics have nothing to do with constant-complement updates, the ideas of information content, and in particular the idea of characterizing database morphisms not by how they map models but rather how they map sentences might prove useful in understanding and characterizing such operations.

Acknowledgment. The core ideas for this research were developed while the author was a visitor at the Information Systems Engineering Group at the University of Kiel. He is indebted to Bernhard Thalheim and the members of the group for the invitation and for the many discussions which led to this work. Also, the anonymous reviewers made numerous suggestions which (hopefully) have led to a more readable presentation.

References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. Database Systems 6, 557–575 (1981)
2. Lechtenbörger, J.: The impact of the constant complement approach towards view updating. In: Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, San Diego, California, June 09-11, 2003, pp. 49–55 (2003)
3. Hegner, S.J.: An order-based theory of updates for database views. Ann. Math. Art. Intell. 40, 63–125 (2004)
4. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Programming Languages and Systems 29 (2007)
5. Hegner, S.J.: Information-optimal reflections of view updates on relational database schemata. In: Hartmann, S., Kern-Isberner, G. (eds.) FoIKS 2008. LNCS, vol. 4932, pp. 112–131. Springer, Heidelberg (2008)
6. Paredaens, J., De Bra, P., Gyssens, M., Van Gucht, D.: The Structure of the Relational Database Model. Springer, Heidelberg (1989)
7. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
8. Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann, San Francisco (1987)
9. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (1990)

10. Jacobs, B.E., Aronson, A.R., Klug, A.C.: On interpretations of relational languages and solutions to the implied constraint problem. ACM Trans. Database Systems 7, 291–315 (1982)
11. Herrlich, H., Strecker, G.E.: Category Theory. Allyn and Bacon (1973)
12. Monk, J.D.: Mathematical Logic. Springer, Heidelberg (1976)
13. Fagin, R.: Horn clauses and database dependencies. J. Assoc. Comp. Mach. 29, 952–985 (1982)
14. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn clauses. J. Logic Programming 3, 267–284 (1984)
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoret. Comput. Sci. 336, 89–124 (2005)
16. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. J. Assoc. Comp. Mach. 31, 718–741 (1984)
17. Hegner, S.J.: Unique complements and decompositions of database schemata. J. Comput. System Sci. 48, 9–57 (1994)
18. Thalheim, B.: Entity-Relationship Modeling. Springer, Heidelberg (2000)
19. Hegner, S.J.: The complexity of embedded axiomatization for a class of closed database views. Ann. Math. Art. Intell. 46, 38–97 (2006)
20. Fagin, R.: Inverting schema mappings. ACM Trans. Database Systems 32 (2007)

A Top-Down Approach to Rewriting Conjunctive Queries Using Views

Nima Mohajerin and Nematollaah Shiri

Department of Computer Science & Software Engineering
Concordia University, Montreal, QC, Canada
{n_mohaje,shiri}@cse.concordia.ca

Abstract. The problem of answering queries using views is concerned with finding answers to a query using only answers to views. In data integration context with the Local-As-Views approach, this problem translates to finding maximally contained rewriting for a given query. Existing solutions follow a bottom-up approach and, for efficiency reason, often require a post-processing phase, which comes at an additional cost.

We propose a solution which follows a top-down approach. For this, we first present a graph-based model for conjunctive queries and views, and identify conditions that if satisfied ensures maximality of a rewriting. Using this model as a basis, we then introduce a novel top-down algorithm, TreeWise, which efficiently generates maximally contained rewritings which are in general less expensive to evaluate, compared to the bottom-up algorithms, without requiring post-processing. The preliminary results of our experiments indicate that while TreeWise has comparable performance, it generally produces better quality rewritings.

1 Introduction

A variety of data-management applications deal with the problem of answering queries using views. Examples include query optimizations, physical data independence, data-warehousing and data integration systems [Lev01]. This problem, also referred to as rewriting query using views, is described as follows: given a query and a set of views over the same database schema, is it possible to find the answers to the query using only answers to the views?

We are interested in the rewriting problem in two contexts. In the context of query optimization and physical data independence [CS95], the goal is to find the rewritings that are equivalent to the original query. In the context of data integration, since finding equivalent rewritings is not always possible, the goal is to find maximally contained rewritings for a given query. We study the problem of answering queries using views following in the context of the LAV-approach in data integration where query as well as views are standard conjunctive queries. Furthermore, in this context we assume that data sources may be incomplete, i.e., they may be missing tuples that satisfy their definitions. This is referred to as the Open-World-Assumption (OWA). There are two challenges in developing algorithms in this context: (1) scalability as the number of views increases, and (2) quality of rewritings generated by the algorithm in terms of cost of evaluation.

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 180–198, 2008.

There are three main algorithms proposed in the literature for producing maximally-contained rewritings in data integration context. These are the Bucket algorithm [LRO96a, LRO96b], inverse-rules [Dus97, Qia96], and the Minicon algorithm [PL00]. In [PL00] authors study the performance limitations of the Bucket and inverse-rule algorithms and showed that Minicon significantly outperforms the two. However, little attention has been paid to the quality of rewritings generated. Following [LMSS95], we consider the quality of a query to be determined by the cost of its evaluation. All three algorithms employ a bottom-up approach to rewriting, i.e., they examine how each subgoal in the body of the query can be covered by some view in the rewriting. This may result in rewritings that have large number of subgoals in the bodies, and hence expensive to evaluate. To solve this problem, both bucket [LRO96a] and Minicon [PL00] propose post-processing steps, at additional cost, for reducing the size of generated rewritings. Our motivation in this work is to explore this problem taking a top-down approach in which, instead of using a view to cover a subgoal of the query, we consider the maximum number of subgoals the view can cover. We provide a graph-based view as a basis to represent the query and views and use it to develop a new rewriting algorithm, called TreeWise, which improves existing algorithms in some aspects. In particular, each rule in a rewriting generated would have less number of subgoals in the body without post-processing, confirmed by preliminary results of our experiments.

The outline of this paper is as follows. Section 2 provides a background. Section 3 presents HMCQ, a hyper-node model for conjunctive queries. In Section 4, we study this model and its connection to rewriting together with conditions that should be satisfied for a contained rewriting to be maximal. This provides a basis for the development of TreeWise algorithm, described in Section 5. A summary of our preliminary experimental results is provided in Section 6. He final section include concluding remarks and future work.

2 Preliminaries

In data integration systems implemented with the Local-As-Views (LAV) approach (e.g., Information Manifold [LRO96b]), source descriptions are defined as views over the mediated schema [Len02]. In this context, the task of answering user's query corresponds to reformulating the original query into a query which only refers to the set of views defined over the mediated schema. Also, in this context, views are not assumed to be complete and they may be missing tuples that satisfy their definitions. Hence, Open-World Assumption (OWA) [AD98] is considered. In this paper, we focus on query and views that are in the form of conjunctive queries, defined as follows.

Definition 1. *(Standard Conjunctive Query) A conjunctive query Q is a non-recursive datalog rule of the form:*

$$q(\overline{X}) :- \quad p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$$

where q and $p_1, \ldots, p_n$ are predicate names, $q(\overline{X})$ is the head, and $p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$ is the body which is a conjunction of atoms, each of which is

called a subgoal. Each arguments in the predicates could be either a variable or constant. The variables in the head arguments $\overline{X}$ *are called* distinguished*; other variables are called* existential*. Union of conjunctive queries is expressed as a set of conjunctive queries with the same head predicate. We denote by* $Q(D)$*, the set of tuples obtained by evaluating* Q *over the input database* D*.*

To compare the answers of two conjunctive queries for any input database, we use the notion of containment. Next, we recall definition of query containment and its necessary and sufficient condition, namely the containment mapping [CM77].

Definition 2. *(Containment Mapping) Given two standard conjunctive queries* Q_1 *and* Q_2*, a c.m.* ρ *from* Q_2 *to* Q_1*, denoted by* $\rho : Q_2 \rightarrow Q_1$*, is a symbol mapping which is identity on the constants and predicate names such that (1)* $head(Q_1) = \rho(head(Q_2))$*, and (2)* $\rho(body(Q_2)) \subseteq body(Q_1)$*. Here* $\rho(head(Q))$ *and* $\rho(body(Q))$ *represent, respectively, the head and body of* Q *after applying* ρ*.*

We say query Q_1 is contained in Q_2, denoted by $Q_1 \sqsubseteq Q_2$, iff there exists a containment mapping from Q_2 to Q_1. Next, we define reformulation of the query also known as rewriting.

Definition 3. *(Rewriting) Given a query* Q *and a set of view definitions* $V = \{V_1, \ldots, V_n\}$*, a rewriting of* Q *using the views in* V *is a query* R *whose ordinary subgoals are all from* V*.*

Since finding all answers to a query in a data integration system is not always possible, contained rewritings are the best we can hope for. In this case, we need to determine a rewriting that returns best possible set of answers to the query. This rewriting is called *maximally contained rewriting*, and is a language dependent notion.

Definition 4. *(Maximally Contained Rewriting) Given a language* L*, query* Q*, and a set of view definitions* $V = \{V_1, \ldots, V_n\}$*, a query* R *is a Maximally contained (MC) rewriting of* Q *using* V *with respect to* L *if:*

1. R *is expressed in* L *and* $R \sqsubseteq Q$*.*
2. $R' \sqsubseteq R$*, for each contained rewriting* R' *of* Q *using* V*, where* R' *is expressed in* L*.*

In [CM77], the problem of containment for conjunctive queries is shown to be NP-complete. In [LMSS95], authors show that the rewriting problem for standard conjunctive queries is NP-complete. Furthermore, it was shown that when query and views are standard conjunctive queries, maximally contained rewriting exists in a form of the union of standard conjunctive queries, where the number of subgoals in each rewriting query in this union will be no more than the number of ordinary subgoals in the original query [LMSS95].

3 Hyper-Node Model for Conjunctive Queries (HMCQ)

We now propose a graph-based model to represent conjunctive queries. We will later use this model to formalize the conditions that must be satisfied in our top-down approach to ensure maximal containment of rewriting. The *hyper-node model* is a graph-based approach proposed in [LP90] intended for representing data models in general, in which nodes can themselves be hyper-nodes . We adopt this model in our work and extend it to *Hyper-node Model for Conjunctive Queries (HMCQ).*

A conjunctive query in HMCQ is a super-graph which consists of four graphs, each representing a level of abstraction. The lowest level in this representation captures relationships among the attributes in the query and possibly equality constraints (i.e., joins) among the attributes. Predicates in the query are represented as hyper-nodes in the second level. The two next levels in HMCQ are used to represent the head of the query.

Let Q be a standard conjunctive query. In HMCQ, Q can be represented using a super-graph called G_Q, which contains the following four graphs.

Definition 5. *(Attributes-graph). The attributes-graph of Q denoted as $G_A = (V_A, E_A)$ is an undirected, node-labeled graph that has the following properties:*

1. *V_A is the set of constants and variables in the body of Q. There is a separate node for each occurrence of an attribute in each predicate regardless of attribute distinctness. This set can be divided into three subsets V_{hc}, V_d, and V_e. V_{hc} includes all the nodes representing constants in the head of Q. V_d contains nodes for distinguished variables and V_e represents the existential attributes in Q. V_e itself is further divided into two disjoint subsets: V_{ev} includes existential variables and V_{ec} includes the constants.*
2. *Set E_A contains the edges between nodes in V_A representing equality relationships among variables in Q. $E = E_d \cup E_e$, where E_d includes the edges between nodes of V_d, and E_e represents the edges between the nodes in V_{ev}.*
3. *Node labels are used to represent orders at different levels of super-graph G_Q. Labeling is performed in such a manner that the node representing attribute 'A' in the j^{th} position in predicate p_i would be labeled (i, j, k, val), where $i \geq 0$ is called the* predicate-index*, which is the unique identifier assigned to predicate p_i in predicates-graph of Q, k is a set called the* head-index *representing the positions of 'A' in the head of Q. The value of k is 0 for elements in V_e. Finally, val is the* value-index*, representing the value of the constant nodes in V_{ec}. Naturally val is empty for the variable nodes.*

Definition 6. *(Predicates-graph). The predicates-graph G_P of Q denoted as $G_P = (V_P, E_P)$ is an undirected, node-labeled, edgeless graph with the following properties:*

1. *V_P is a set of hyper-nodes, each of which representing a predicate p_i in the body of Q and each containing a set of nodes from the graph G_A, representing the arguments of p_i in Q.*

2. *For the hyper-node representing predicate p_i in Q, we use $(i, name)$ as the label, where $i \geq 0$ is a unique identifier assigned to predicate p_i indicating the i^{th} subgoal in the body, and "name" is the predicate name.*

Definition 7. *(Head-variables-graph). This graph of Q, denoted as $G_{HV} = (V_{HV}, E_{HV})$, is an undirected, edgeless graph, where V_{HV} is a set of hyper-nodes, each of which representing an attribute of the head predicate. Each element in this set is either a distinguished variable or a constant. Therefore V_{HV} contains either a connected component from V_d or an element in set V_{hc} of graph G_A.*

Definition 8. *(Head-graph). The head-graph of Q, denoted as $G_H = (V_H, E_H)$, is an undirected, edgeless graph, where V_H is a set which contains a hyper-node representing the head predicate of the conjunctive query. This hyper-node itself contains all the hyper-nodes of graph G_{HV}.*

Since the essential information of G_H and G_{HV} are captured in labeling of the attribute-nodes, for the sake of clarity, we sometimes omit these graphs from our representations in HMCQ.

Example 1. Consider the following conjunctive queries:

$$Q1(X, Y, Z) :- \quad p(X, Y, Z), r(X, U, V).$$
$$Q2(X, Y, Z) :- \quad p(X, Y, X), r(X, Y, Z).$$

In the HMCQ model, $Q1$ and $Q2$ can be represented as shown in Fig. 1. In the graph of $Q1$, hyper-nodes of the head-variables graph and head-graph are shown, however in the graph of Q2, these are omitted for sake of clarity. If an attribute node does not belong to more than one head variable in the head-variables-graph of the query, we can use an integer index to represent its position in the head in its label instead of a set of integers. This is followed in the graph of $Q2$.

It is noteworthy that related concepts such as containment-mapping, containment, and rewriting queries, and unfolding technique have corresponding definitions in the HMCQ model. However, due to space limitations, we suppress some

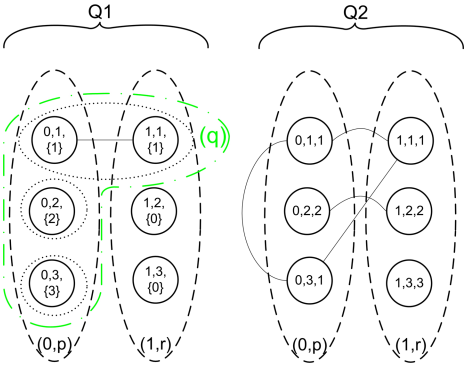


Fig. 1. Graphs of queries $Q1$ and $Q2$ for Example 1

details in this work. Also, for every standard conjunctive query Q, there exists a unique super-graph G_Q in HMCQ representing Q. We use $\mathbf{Graph(Q)}$ to denote a function which returns this unique graph G_Q of Q. Interestingly, this function is 1-1 (trivial variable renaming may be needed) and therefore $\mathbf{Graph^{-1}(G_Q)}$ returns Q.

4 A Top-Down Approach to Rewriting Using HMCQ

Using the HMCQ model, in this section we present a top-down approach to generate rewriting for standard case of conjunctive query and views. We study the conditions that must be satisfied by HMCQ to ensure maximality of rewriting.

Our top-down approach to the rewriting problem includes the following three phases: establishing consistent partial mappings from the query to each view, examining partial mappings to ensure maximality of rewriting, and finally combining the partial mappings properly to generate maximally contained rewriting in the form of union of standard conjunctive queries. For ease of exposition and also conforming to Minicon description in [PL00], we assume the query and views do not include constant arguments. Details of these phases are described as follows.

4.1 Generating Partial Mappings

In the first phase of our top-down approach, we examine each view V_i, in isolation, and construct a set of consistent partial mappings from the query to V_i, each of which covering maximal number of subgoals of the query. Next, we discuss the details of this phase using the HMCQ model.

Let G_Q be the super-graph representing the query Q, and G_{V_i} be the one representing a view in the set V of views. In general, a desired partial mapping must satisfy the following four conditions to ensure containment and maximality of a rewriting: (1) head-unification, (2) join-recoverability, (3) partial-mapping-consistency, and (4) partial-mapping-maximality.

The first three conditions above govern consistency of the mappings and containment of each rewriting query r_j induced by them. The last condition ensures maximality of a rewriting induced by these partial mappings. We next describe these conditions for a partial mapping μ_j in the HMCQ model. In the following conditions, D_{μ_j} is a set representing the domain of μ_j.

Head-unification condition: Testing this condition is straightforward. To satisfy this condition in HMCQ for a partial mapping μ_j from G_Q to G_{V_i}, the following must hold:

$$\forall n_k \in (V_d)_Q : \qquad n_k \in D_{\mu_j} \Rightarrow \mu_j(n_k) \in (V_d)_{V_i} \tag{1}$$

The above indicates that if a distinguished node in G_Q is mapped to an existential node in G_{V_i}, this condition is violated. This is also consistent with the definition of containment mapping, which implied that all the distinguished

variables of the containing query (Q) must be mapped only to the distinguished variables in the containee (e.g. a rewriting query r).

Join-recoverability condition: The following must hold for all existential edges $e =< n_k, n_{k'} >$ in $(E_e)_Q$ in order to satisfy this condition:

$$\begin{array}{l} \forall e =< n_k, n_{k'} > \in (E_e)_Q : \\ (n_k \in D_{\mu_j} \wedge n_{k'} \notin D_{\mu_j}) \Rightarrow \mu_j(n_k) \in (V_d)_{V_i} \end{array} \tag{2}$$

This condition focuses on the existential edges in the attributes-graph of G_Q, where one of the nodes, but not both, is in the domain of μ_j. In order to satisfy this condition, the node in the domain of μ_j must be mapped to a distinguished node in G_{V_i}.

Partial-mapping-consistency condition: Let H be a subset of all possible edges that can be created between distinguished nodes in attributes-graph of the view V_i, that is $H \subseteq (V_d)_{V_i} \times (V_d)_{V_i}$. To satisfy partial-mapping-consistency condition, the following must hold for μ_j:

$$\begin{array}{l} \exists H : \forall < n_k, n_{k'} > \in (E_A)_Q : \\ n_k, n_{k'} \in D_{\mu_j} \Rightarrow < \mu_j(n_k), \mu_j(n_{k'}) > \in (H \cup (E_A)_{V_i}) \end{array} \tag{3}$$

Here, we are concerned with every edge $< n_k, n_{k'} >$ in G_Q connecting the nodes that both are in the domain of μ_j. To satisfy this condition, mapping of all such edges must either exist in G_{V_i} or can be created in the graph of the view. In case both $\mu_j(n_k)$ and $\mu_(n_{k'})$ are in $(V_e)_{V_i}$, presence of edge $< \mu_j(n_k), \mu_j(n_{k'}) >$ in $(E_e)_{V_i}$ indicates that there is a dependency between the hyper-nodes p_n and $p_{n'}$ with respect to μ_j, where $n_k \in p_n$ and $n_{k'} \in p_{n'}$. That is, removal of the attributes of any one (but not both) of these predicates from domain of μ_j will cause violation of join-recoverability condition (condition 2) by the attribute nodes of the other predicate.

Example 2. Consider the following query and views:

$$\begin{array}{ll} Q(X,Y,W) :- & p(X,Y,Z), r(Z,U,U), s(W,Y). \\ V1(A,B,D,E,F) :- & p(A,B,C), r(C,D,A), s(E,F). \\ V2(A,B,D) :- & p(A,B,C), s(A,D). \end{array}$$

Fig. 2 shows the super-graphs of Q, $V1$, and $V2$. The only maximal mapping from G_Q to G_{V1} is $\mu_1 = \{(0,1,1) \rightarrow (0,1,1), (0,2,2) \rightarrow (0,2,2), (0,3,0) \rightarrow (0,3,0), (1,1,0) \rightarrow (1,1,0), (1,2,0) \rightarrow (1,2,3), (1,3,0) \rightarrow (1,3,1), (2,1,3) \rightarrow (2,1,4), (2,2,2) \rightarrow (2,2,5)\}$

This mapping covers the hyper-nodes $\{(0,p), (1,r), (2,s)\}$ in G_Q. We now check μ_1 for the above three conditions.

- *head-unification:* The subset of $(V_d)_Q$ in the domain of μ_1 is $\{(0,1,1), (0,2,2), (2,1,3), (2,1,3), (2,2,2)\}$. Since every node in this set is mapped to an element in $(V_d)_{V1}$, none of the mappings violate head-unification condition.
- *join-recoverability:* Since all the nodes in set, $(E_e)_Q = \{< (0,3,0), (1,1,0) >, < (1,2,0), (1,3,0) >\}$ are in the domain of μ_1, this condition is satisfied.
- *partial-mapping-consistency:* For the set $(E_A)_Q = \{< (0,3,0), (1,1,0) >, < (1,2,0), (1,3,0), < (0,2,2), (2,2,2) >\}$, we have the following:

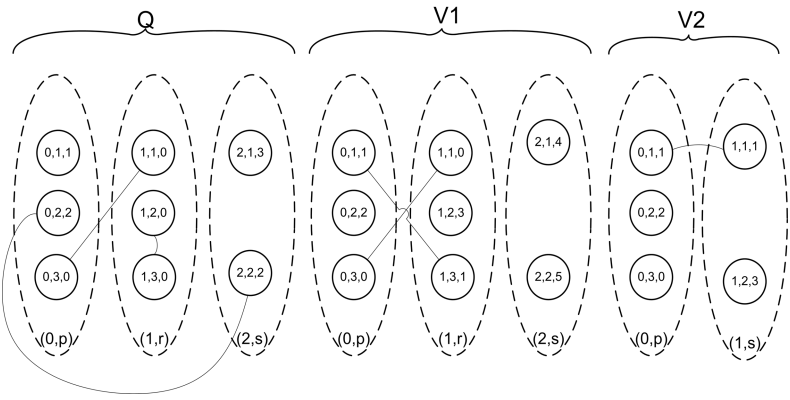


Fig. 2. Super-graphs of Q, V1, and V2 in Example 2

- according to μ_1, the source and destination nodes of edge $< (0,3,0),(1,1,0) >$ are mapped to $(0,3,0)$ and $(1,1,0)$, respectively. Since both of these nodes belong to set $(V_e)_{V1}$, the edge $< (0,3,0),(1,1,0) >$ must be in $(E_e)_{V1}$, which is indeed true. Additionally, this edge creates a dependency, in the context of μ_1, between the hyper-nodes $(0,p)$ and $(1,r)$ of G_Q for μ_1.
- according to μ_1, the nodes of edge $e =< (1,2,0),(1,3,0) >$ are mapped to $(1,2,3)$ and $(1,3,1)$ respectively, both of which belong to set $(V_d)_{V1}$. The edge e can thus be added to set H, and therefore condition 3 is not violated.
- Since edge $< (0,2,2),(2,2,2) >$ belongs to $(E_d)_Q$, according to condition 1, both nodes of this edge will be mapped to distinguished nodes in $(V_d)_{V1}$. These nodes are mapped to $(0,2,2)$ and $(2,2,5)$, respectively. The edge does not exist in $(E_d)_{V1}$, but it can be added and therefore condition 3 is not violated.

At this point, the first three conditions are satisfied in the mapping μ_1. Similarly, for $V2$ we have the partial mapping $\mu_1' = \{0,1,1) \rightarrow (0,1,1), (0,2,2) \rightarrow (0,2,2), (0,3,0) \rightarrow (0,3,0), (2,1,3) \rightarrow (1,1,1), (2,2,2) \rightarrow (1,2,3)\}$, which also satisfies all three conditions mentioned above and covers the set $\{(0,p),(2,s)\}$ from G_Q.

Lemma 1. *Let Q be a query and V be a set of views all in the standard conjunctive form. Let T be the set of partial mappings from the super-graph G_Q to those for views, and G_R be the rewriting induced by T. If the query $Graph^{-1}(G_R)$ is contained in Q, then every element in T satisfies head-unification, join-recoverability, and partial-mapping-consistency conditions.* □

Partial-mapping-maximality condition: In order to ensure that a partial mapping μ_j would produce maximally contained rewriting, we need to closely examine the edges between the attributes of the view in the range of μ_j. This is to ensure that no extra constraints are added to the rewriting unless it is necessary.

It was mentioned earlier that the partial-mapping-consistency condition may sometimes create dependencies between predicates of the query that are in the domain of μ_j. As a result, the hyper-nodes in the predicates-graph of query in domain of μ_j are partitioned into a set of connected components, where the predicates in each component can not be removed from the mapping without violation of the join-recoverability condition by the remaining predicates of that component. We refer to this set of connected components in the domain of μ_j as C_{μ_j}. Before describing this condition, we introduce a few concepts in HMCQ, used in our formalization.

- $Pred(G_Q, n)$ is a function from $(V_A)_Q$ to $(V_P)_Q$, which returns for each node $n \in (V_A)_Q$ the hyper-node $p \in (V_P)_Q$, where $n \in p$.
- For partial mapping μ_j, we use μ_j^{-1} to denote the inverse of μ_j, defined in the usual way.
- $Comp(C_{\mu_j}, p)$ is a function which returns the component c in C_{μ_j} to which the predicate hyper-node p belongs.

For Partial-mapping-maximality condition we focus on the set of edges that a view G_{V_i} enforces on to the attribute nodes of G_Q induced by a mapping μ_j. This set of edges is formalized as follows:

Definition 9. *Let S be the set of all edges that are enforced by G_{V_i} through the partial mapping μ_j. This set includes the following elements:*

- *The collection of sets $\mu_j^{-1}(e_k)$, for every edge $e_k =< n, n' > \in (E_A)_{V_i}$ where both n and n' are in the range of μ_j. Note the abuse of notation: we use $\mu_j^{-1}(e_k)$ to refer to all possible edges between sets $\mu_j^{-1}(n)$ and $\mu_j^{-1}(n')$ for $e_k =< n, n' >$.*
- *For set P of predicate hyper-nodes in G_Q that are mapped to the same predicate hyper-node $p_{k'}$ of G_{V_i}, edges created between every pair of nodes from set P that have the same position-index in their labels.*

Using S and also considering the set of components C_{μ_j},
partial-mapping-maximality is satisfied if the following condition holds:

$$\forall < n_k, n_{k'} > \in S : Comp(C_{\mu_j}, Pred(G_Q, n_k)) \neq Comp(C_{\mu_j}, Pred(G_Q, n_{k'})) \Rightarrow \quad < n_k, n_{k'} > \in (E_A)_Q \tag{4}$$

The above indicates that for each edge $< n_k, n_{k'} >$ in S, where predicates for the nodes of the edges do not belong to the same components in C_{μ_j}, if $< n_k, n_{k'} >$ is not in $(E_A)_Q$, then this condition is violated.

Example 3. Consider the partial mappings μ_1 and μ_1' for the query and views in Example 2. We now can test these two mapping for partial-mapping-maximality. For $\mu_1 = \{(0,1,1) \rightarrow (0,1,1), (0,2,2) \rightarrow (0,2,2), (0,3,0) \rightarrow (0,3,0), (1,1,0) \rightarrow (1,1,0), (1,2,0) \rightarrow (1,2,3), (1,3,0) \rightarrow (1,3,1), (2,1,3) \rightarrow (2,1,4), (2,2,2) \rightarrow (2,2,5)\}$, we have $S_{\mu_1} = \{< (0,1,1), (1,3,0) >, < (0,3,0), (1,1,0) >\}$ and $C_{\mu_1} = \{\{(0,p), (1,r)\}, \{(2,s)\}\}$. It can be seen, $< (0,1,1), (1,3,0) >$ is the

only edge in S that is not present in $(E_A)_Q$. However, since $Comp(C_{\mu_1}, (0,p)) = Comp(C_{\mu_1}, (1,r))$, condition 4 is not violated by μ_1.

Now let us consider the mapping $\mu_1' = \{(0,1,1) \rightarrow (0,1,1), (0,2,2) \rightarrow (0,2,2), (0,3,0) \rightarrow (0,3,0), (2,1,3) \rightarrow (1,1,1), (2,2,2) \rightarrow (1,2,3)\}$ for $V2$. We have: $S_{\mu_1'} = \{< (0,1,1), (2,1,3) >\}$ and $C_{\mu_2} = \{\{(0,p)\}, \{(2,s)\}\}$. Here, the edge $< (0,1,1), (2,1,3) >$ is not in $(E_A)_Q$. Since $Comp(C_{\mu_2}, (0,p)) \neq Comp(C_{\mu_2}, (2,s))$, condition 4 is violated.

Lemma 2. *Let Q and v be a query and a view in standard conjunctive form. If G_R is a contained rewriting of G_Q generated from set T of partial mappings with disjoint subgoal coverage from G_Q to G_v, where each element in T satisfies partial-mapping-maximality condition, then $Graph^{-1}(G_R)$ is maximally contained in Q.* □

4.2 Partial Mappings and the Impact of OWA

So far, we described the conditions for generating consistent partial mappings which can induce contained rewriting queries for Q (and if there exists only one view, the generated rewriting will be maximally contained). However, there is an issue involved in our top-down approach, which we need to address to ensure maximality of result rewriting when multiple views exist. This issue is related to Open-World Assumption (OWA), which implies that partial mappings from different views which cover the same subgoals must also be broken down before combined together to guarantee maximally contained rewriting.

To address this issue, we consider an additional phase in our top-down approach, in which we compare the partial mappings. If partial mappings are from different views, when their subgoal coverage are not disjoint, we must separate disjoint portions of the mappings from which we create new mappings. This condition is formalized next, for which we need to introduce some terms as follows.

- $View(\mu_j)$ is a function that returns the view for which the μ_j is defined.
- $Subgoals(\mu_j)$ denotes the maximal set of predicate hyper-nodes of the query in the domain of μ_j. Similarly, for a set M of partial mappings, $Subgoals(M)$ returns the union of maximal predicate hyper-nodes in the domain of each partial mapping μ_k in M.
- $Comps(\mu_j)$ is the set of components for subgoals of the query in the domain of μ_j. As mentioned earlier, these components represent dependencies between the subgoals with respect to μ_j.
- Subset relationship ($\mu_1 \subseteq \mu_2$) between two partial mappings μ_1 and μ_2 (from the same view) is defined as μ_1 being an extension of μ_2 and with $Subgoals(\mu_1) \subseteq Subgoals(\mu_2)$.
- Equality relationship ($\mu_1 = \mu_2$) between two partial mappings μ_1 and μ_2 means that $\mu_1 \subseteq \mu_2$ and $\mu_2 \subseteq \mu_1$.

Complete-mapping-maximality condition: Let M be the set of partial mappings generated for a query Q and a set of views V. Complete-mapping-maximality condition is satisfied if the following holds.

$$\forall \mu_i \in M : (\exists \mu_j \in M : View(\mu_i) \neq View(\mu_j) \wedge Subgoals(\mu_i) \cap Subgoals(\mu_j) = S) \\ \Rightarrow \exists M' \subseteq M : (\forall \mu'_k \in M' : \mu'_k \subseteq \mu_i \wedge |Comps(\mu'_k)| = 1) \wedge Subgoals(M') = S \tag{5}$$

The above condition states that for every partial mapping μ_i in M generated during the first phase, if there exists some other partial mapping μ_j in M from a different view where the two mappings share subgoal coverage, then there should exist a subset M' of the partial mappings in M such that each mapping μ'_k in M' is a subset of μ_i and the subgoal coverage of μ'_k is a component in the intersection of μ_i and μ_j. Simply put, for every two partial mappings from different views, their overlapping subgoal coverage must be created into minimal components.

Example 4. Suppose set M contains the mappings μ_1, μ'_2 and μ'_3 for the query and views in Example 3. Recall that for μ_1, $\{\{(0,p),(1,r)\},\{(2,s)\}\}$ is the coverage of the mapping in form of components. As was shown in the example, μ_2 violates condition 4, and hence we replace it with μ'_2 and μ'_3 having the coverage $\{\{(0,p)\}\}$ and $\{\{(2,s)\}\}$, respectively.

For μ_1, we notice that this mapping overlaps with μ'_2 over $(0,p)$. Therefore, an extension of μ_1 must exist in M having coverage equal to component to which $(0,p)$ belongs. However, no such mapping exists in M and therefore this condition is violated. To rectify this, we can add a new mapping μ_{11} to M with coverage $\{\{(0,p),(1,r)\}\}$. By comparing the mappings μ'_3 and μ_1, we note that the two share the predicate $(2,s)$, and since no such extension of μ_1 exists in M, again this condition is violated. To rectify this, we need to add μ_{12} with coverage $\{(2,s)\}$ to M. For mappings μ'_2 and μ'_3, we notice that condition 5 is satisfied. Hence, the result of comparing the mappings is the set $M = \{\mu_1, \mu_{11}, \mu_{12}, \mu'_2, \mu'_3\}$, where the subgoal coverage of μ_1 is $\{\{(0,p),(1,r)\},\{(2,s)\}\}$, μ_{11} is $\{\{(0,p),(1,r)\}\}$, μ_{12} is $\{\{(2,s)\}\}$, μ'_2 is $\{\{(0,p)\}\}$, and the subgoal coverage of μ'_3 is $\{\{(2,s)\}\}$.

4.3 Rewriting Generation in HMCQ

The final phase in our top-down approach to rewriting focuses on generation of rewriting R using the partial mappings in M. The main task in our rewriting generation phase is to efficiently combine partial mappings in M and generate a set of complete mappings, each of which resulting in a conjunctive query covering the entire body of Q. Union of these conjunctive queries form a maximally contained rewriting of Q using V. In doing so, we take advantage of the following property.

Property 1. When combining partial mapping from M to generate rewriting for G_Q using a set graphs of views in V, for every combination C that results in a non-redundant rewriting, the following two conditions must hold:

1. $\forall \mu_i \& \mu_j \in C : Subgoals(\mu_i) \cap Subgoals(\mu_j) = \emptyset$
2. $\nexists \mu' \in M : (\exists \mu_1 \ldots \mu_k \in C : (\mu_1 \cup \mu_2 \cup \ldots \cup \mu_k = \mu'))$

Using these two properties, we can eliminate many useless combinations of partial mappings in the rewriting phase. The first condition in Property 1 is the same as disjoint property in Minicon [PL00]. The second condition in Property 1 is unique to our approach and can be used to further reduce the search space in the rewriting generation phase. The following example illustrates these points.

Example 5. Let set M be a set of mappings generated for query Q and views V_1 and V_2 in Example 2. To generate a rewriting R for Q, we examine combinations of those elements in M which satisfy both conditions of the above property as follows. The only combinations that satisfy both conditions are $\{\mu_1\}$ and $\{\mu_{11}, \mu'_3\}$. These queries are as follows:

$$R1(X, Y, W) :- \quad V1(X, Y, X, W, Y).$$
$$R2(X, Y, W) :- \quad V1(X, Y, X, E_1, F_1), V2(W, B_1, Y).$$

It is noteworthy that for this example, the Minicon algorithm produces the following rewriting R'.

$$R'1(X, Y, W) :- \quad V1(X, Y, X, E_1, F_1), V1(A_1, B_1, D_1, W, Y).$$
$$R'2(X, Y, W) :- \quad V1(X, Y, X, E_2, F_2), V2(W, B_2, Y).$$

5 TreeWise Rewriting Algorithm

We can now present details of our top-down algorithm, called TreeWise, which uses the HMCQ model. By using the conditions and properties captured in HMCQ and by carefully choosing proper data structures and procedures, TreeWise is set to efficiently generate better quality rewritings without any post-processing.

The algorithm operates in three phases: mapping tuple construction, binary-tree construction, and rewriting generation phase. The first phase corresponds to the generation of partial mappings presented using HMCQ. In the binary tree construction phase, we address the issues involved in partial mapping comparison discussed in our analysis. Finally, the rewriting generation phase includes steps for combining partial mappings and generation of rewritings. In what follows, we describe details of three phases.

5.1 Mapping Tuple Construction Phase

In section 4.1, we described the four necessary conditions to ensure usability of partial mappings in generating maximally contained rewriting. We now use this knowledge to generate a set of consistent partial mappings each of which satisfying these conditions. In this phase of the TreeWise Algorithm, for each view V_i in V, a set of mapping tuples T_{V_i} is created. Each element $t =< \mu_t, \rho_t, ((G_H)_{V_i})_t, ((G_P)_Q)_t, C_t, Subs_t >$ in T_{V_i} is defined as follows.

- μ_t is a partial mapping from a subset of attribute-nodes of G_Q to nodes of G_{V_i}.
- ρ_t is the conjunctive equivalent of the partial mapping μ_t. That is, ρ_t is a partial mapping from Q to V_i.
- $((G_H)_{V_i})_t$ is a copy of the head-graph of view in HMCQ representing μ_t. To this graph, some edges may be added to make the mapping consistent.
- $((G_P)_Q)_t$ is the copy of the predicates-graph of the query that has edges added during the mapping construction phase to reflect subgoal dependencies in the domain of μ_t (refer to partial-mapping-consistency condition).
- C_t is the set of connected components in $((G_P)_Q)_t$.
- $Subs_t$ is the set of subgoal hyper-nodes in predicates-graph of the query covered by μ_t. That is, $Subs_t = Subgoals(\mu_t)$.

The predicate hyper-nodes in G_{V_i} form a set of equivalence classes, each of which representing targets for a predicate hyper-node in G_Q . Next, for each partial mapping μ_t from G_Q to G_{V_i}, we create a mapping tuple and examine it for the four conditions described in section 4.1.

Head-unification condition: Conforming to condition 1, we check the set V_d of nodes in attributes-graph of the query in the domain of μ_t. For each node n_k in $(V_d)_Q$ that is mapped to an existential node in $(V_e)_{V_i}$, we remove from the domain of μ_t, attributes of hyper-node $p_m \in (V_p)_Q$, where $n_k \in p_m$.

Join-recoverability condition: To satisfy condition 2, for each existential edge $< n_k, n_{k'} >$ in the attributes-graph of the query, where only n_k is in the domain of μ_t, we verify if n_k is mapped to a distinguished node in the view. Otherwise, we have to remove from domain of μ_t, attributes of hyper-node p_m in $(E_P)_Q$, where $n_k \in p_m$.

Partial-mapping-consistency condition: For condition 3 to hold, we have to examine each edge $< n_k, n_{k'} >$ in attributes-graph of the query, where n_k and $n_{k'}$ are both in the domain of μ_t, and find a set H of distinguished edges to add to the view in order to make the mapping consistent. For the three possible cases mentioned in the description of condition 3, TreeWise proceeds as follows:

1. If n_k and $n_{k'}$ are both mapped to existential nodes in the view, then edge $< \mu_t(n_k), \mu_t(n_{k'}) >$ must exist in $(E_A)_{V_i}$. If not, then we remove from the domain of μ_t, nodes of hyper-nodes p_m and $p_{m'}$ in $(E_P)_Q$, where n_k and $n_{k'}$ are in $p_{m'}$ and $p_{m'}$, respectively. However, if the mapping exists in the view, then subgoal dependency exists between p_m and $p_{m'}$ and this dependency is captured by TreeWise in the form of an edge $< p_m, p_{m'} >$ in the copy $((G_P)_Q)_t$ of the predicates-graph of the query for μ_t. After addition of a new edge to $(((G_P)_Q)_t)$, this graph has to be replaced by its transitive closure graph, since dependency relation entails transitivity.
2. If n_k is mapped to an existential node and $n_{k'}$ to a distinguished node, then we have to remove the nodes of p_m from domain of μ_t, where $n_k \in p_m$.
3. If n_k and $n_{k'}$ are mapped to distinguished nodes in V_i, even if the mapping of the edge does not exist in the view, we can add it to the rewriting. Hence, if $\mu_t(n_k)$ and $\mu_t(n_{k'})$ belong to different hyper-nodes in head-graph $((G_H)_{V_i})_t$ of the view and the edge does not already exist, we add it.

After checking every edge and taking the appropriate action, tuple t will now have the set of connected components C_t where the subgoals of each component is dependent on one another.

Partial-mapping-maximality condition: Violation of condition 4 indicates that the current mapping as a whole may not generate maximally contained rewriting. Therefore, in this case, we should replace the mapping by its smaller constituencies, i.e, breaking the mapping into smaller pieces.

The focus of our attention is now on set S presented in Definition 9. Additionally, to decide the breaking strategy, the algorithm keeps track of violations of condition 4 in the form of conflicts between components in predicates-graph of the mapping tuples in the following manner. For every edge e_i in S with nodes belonging to different components of C_t, we test whether e_i is present in

the graph of Q. If not, a conflict pair between predicate hyper-nodes containing nodes of e_i is created. Using dependency in $((G_P)_Q)_t$ and the set containing conflict pairs, TreeWise breaks μ_t into smaller mappings in such a way that each new mapping will have maximum number of components without any conflict between its predicate hyper-nodes. The following is the description of mapping tuple construction phase of the TreeWise algorithm.

procedure **constructMappingTuples**(Q, V, G_Q, G_V)
Inputs: /* Q and V are conjunctive queries in standard form */
/* G_Q and G_V are the graphs of Q and V in the HMCQ model*/
Output: T is a set of consistent mapping tuples.
$T = \emptyset$.
for each view v in V
$T_v = \emptyset$.
Form set E_v of Equivalent Classes of v for subgoals in Q.
for each element e in the cartesian product of the elements in E_v,
where e covers set g of subgoals in Q
Let h be the least restrictive head-homomorphism
on v such that there exists a mapping ρ_t and
its corresponding μ_t in HMCQ such that $\rho_t(Subs_t) = h(e')$, where
$Subs_t \subseteq g$ and $e' \subseteq e$, and μ_t satisfies conditions 1–3.
if h and ρ_t exist, then:
form tuple $t = (\mu_t, \rho_t, v, (G_H)_v, ((G_P)_Q)_t, C_t, Subs_t)$, where:
a) $(G_H)_v$ is the head-graph of v with
with minimal set of edges to represent h.
b) $((G_P)_Q)_t$ is the predicates-graph of Q with
minimal set of edges to represent dependency from
condition 3.
c) C_t are the connected components in $((G_P)_Q)_t$
$Conflicts = \emptyset$.
Form set S_t described in Definition 9 for tuple t.
for each edge $< n_1, n_2 >$ in S_t where
$Comp(C_t, Pred(G_Q, n_1)) \neq Comp(C_t, Pred(G_Q, n_2))$:
if $< n_1, n_2 >$ is not in G_Q, then:
add pair $(Pred(n_1), Pred(n_2))$ to $Conflicts$.
Break t into minimal set of mapping tuples T_t such that for each
element
t' in T_t, $Subs_{t'}$ does not contain two elements p_1 and
p_2 where (p_1, p_2) is in $Conflicts$.
$T_v := T_v \cup T_t$.
$T := T_v \cup T$.
Return T

5.2 Binary Tree Construction Phase

During its second phase, the algorithm checks for complete-mapping-maximality condition. However, in case of violation, the partial mapping does not have to

be removed from the set. Instead, new smaller mappings that are all extensions of the original mapping must be created and added to the set of mapping tuples. We refer to this process as *cloning* a partial mapping tuple into *children* mapping tuples. To facilitate testing of condition 5 and also to keep track of parent-child relationships between the mapping tuples, the TreeWise algorithm uses *binary trees* (hence the name TreeWise) in the following manner.

For each partial mapping tuple generated in the previous phase, TreeWise creates a binary tree with this tuple as its root. Therefore set T of mapping-tuples becomes a set T' of binary trees. Next, the root of each binary tree t'_i in T' is compared with the leaf-tuples of all other trees in T' that are from different view (than the root). For each leaf-tuple m_{j_k} of t'_j that has intersection on subgoal coverage with the root of t'_i, one intersection and one difference tuple are generated and added as children of m_{j_k} in t'_j. Dependencies presented in predicates-graph $((G_P)_Q)$ of m_{j_k} adds, to the intersection child, all the predicates in the components of the intersection.

After comparing roots of every tree with leaves of all other trees that are from different views, condition complete-mapping-maximality may not still be satisfied. We now need to clone the leaf-nodes that have intersection with at least one other tuple into children nodes having only one component based on set C_t of each tuple. With the TreeWise algorithm, this task is straightforward due to using binary trees. The following describes details of the second phase.

procedure **binaryTreeConstruction**(T)
Inputs: /* T is the set of partial mapping tuples
created in the first phase of the algorithm */
Output: T' is a set of binary trees with mapping-tuples as nodes.
$T' = \emptyset$.
for each tuple t in T:
Form a binary-tree t' with t as the root and Add it to T'.
for each root r of tree t' in T':
for each tree t'' in T' such that there exists a leaf-node m in t'' such that $Subs_m$ belongs to more than one component in $((G_P)_Q)_m$:
for each leaf-node m in t'':
if $Subs_m \cap Subs_r \neq \emptyset$ then:
a) form new tuple m_1 such that $Subs_{m_1}$ contains all the predicates hyper-nodes in the components from C_m that have a predicate included in $Subs_m \cap Subs_r$.
b) form new tuple m_2 such that $Subs_{m_2} = Subs_m - Subs_{m_1}$.
Set m_1 as the left child and m_2 as the right child of m.
for each tree t' in T' such that there exists a leaf-node m in t' such that $Subs_m$ belongs to more than one component in C_m:

for each leaf-node m in t' such that $Subs_m$
belongs to more than one component in C_m and $Subs_m$
has intersection with at least one root of a tree in T' :
a) form new child tuple m_1 of m such that $|C_{m_1}| = 1$.
a) form new child tuple m_2 of m such that $Subs_{m_2} = Subs_m - Subs_{m_1}$.
set m_1 as the left child and m_2 as the right child of m.
Return T'

5.3 Rewriting Generation Phase

In the last phase of TreeWise algorithm, rewriting queries are generated from partial mapping tuples nodes of the trees in set T'. Using Property 1, TreeWise algorithm considerably reduces the search space of this phase. For each valid combination c of tuples satisfying Property 1, a rewriting query is generated.

For each tuple t of view v in a combination c, generating rewriting involves creating a subgoal $v(\overline{Y'})$ representing t in the body of the rewriting. For generating subgoal $v(\overline{Y'})$ from view definition $v(\overline{Y})$, we use the partial mapping ρ_n^{-1} to *unmap* distinguished variables ($\overline{Y}$) of the view to variables of the query. There are two issues involved in this task; First of all, for any variable y in $\overline{Y}$ that is not in the range of the partial mapping ρ_n, we must create fresh copy of y and add it to $v(\overline{Y'})$. Secondly, since in ρ_n a set of variables of the query may be mapped to a single variable in $\overline{Y}$ of the view, we define a representative of this set arbitrarily, except we use distinguished variables of query whenever possible. Since this choosing of the representative must be performed uniformly across the tuples of combination c, similar to Minicon algorithm [PL00], we use $EC(\overline{Y})$ to refer to this uniformity across the body of the rewriting.

procedure **rewritingGeneration**(T')
$R = \emptyset$.
for each combination $c = \{t_1, \ldots, t_k\}$ of tuples of tree-nodes in set T'
where:
a) $Subs(t_1) \cup \ldots \cup Subs(t_k)$ cover exactly the subgoals of Q.
b) for any $i \neq j$, $Subs(t_i) \cap Subs(t_j) = \emptyset$.
c) for any $i \neq j$, $parent(t_i) \neq parent(t_j)$.
for each tuple t_i in the combination c:
Define mapping ϕ_i on the $\overline{Y}_i = Graph^{-1}(Head_{t_i})$ as follows:
if variable $y \in \overline{Y}_i$ is in the range of ρ_{t_i}
$\phi_i = \rho_{t_i}^{-1}(y)$.
$\phi_i(y)$ is a fresh copy of y.
Form conjunctive rewriting r:
$r(EC(head(Q))) :- \quad V_{n_1}(EC(\phi_1(\overline{Y}_1))), \ldots, V_{n_k}(EC(\phi_k(\overline{Y}_k)))$.
add r to R.
Return R

6 Experiments and Preliminary Results

Due to space limitations, in this section we only present a summary of the results of our experiments. These experiments were intended to evaluate the performance of TreeWise by comparing it with three versions of the Minicon algorithm which we have implemented. The results are used to analyze and compare strengths and limitations of these algorithms.Our experiments includes three classes of queries: chain, star, and complete queries, as used in [PL00]. In addition to these queries, we also tested the algorithms using conjunctive queries, chosen randomly from a pool of such queries.

To make the results of our experiments comparable to Minicon, we used the same random query generator used in [PL00], which was kindly provided to us by Dr. Pottinger. The results are compared using the following parameters: (1) performance, i.e., the elapsed time to generate the rewriting, (2) the length of the rewriting, i.e., number of rules, (3) the width of the rewriting queries, and (4) the area of the rewriting (the product of query width and length.

Our experiments indicate that in general performance of Minicon and the TreeWise algorithm are very close. In few cases Minicon slightly outperforms TreeWise (i.e., case of star queries where all of the joined variables are distinguished). In some cases, the top-down, view-based approach of TreeWise algorithm outperforms the bottom-up, subgoal-based approach of Minicon (i.e., case of chain queries with only 2 distinguished variables, case of star queries with all non-joined variables being distinguished). We made the following observations from our experiments:

(1) Naturally, a top-down approach produces fewer mapping-tuples during the tuple construction phase, compared to the MCDs generated by Minicon during its first phase. In the second phase, TreeWise generates new tuples only if necessary. Furthermore, the tree structure which keeps track of child-parent relationships improves efficiency of the algorithm in the third phase. (2) When there is a large number of joins between existential variables (case 1 of chain, star, and complete queries), TreeWise with its graph-based approach in recording dependencies between subgoals runs increasingly faster than Minicon algorithm, which discovers these dependencies multiple times. (3) The overhead of length and width optimization routines for Minicon algorithm, when performed, is quite high. In general, the overhead of length optimization grows rapidly in the number of MCDs. In cases where the number of queries in the rewriting is high (e.g., all variables in the query and views are distinguished), the overhead of width optimization is almost exponential.

7 Conclusions and Future Directions

Our analysis of existing algorithms shows that rewritings generated by those algorithms are, in general, expensive to evaluate. This is due to their bottom-up and subgoal-based approach to rewriting. In this paper, our focus was to explore the top-down approach to generating rewriting. We presented TreeWise,

an efficient top-down algorithm which takes measures to satisfy the conditions specified using the HMCQ model. Results of our experiments showed that TreeWise generally produces rewritings that are less expensive to evaluate without requiring post-processing step like Minicon.

As a future work, we plan to complete the experiments to further study quality and scalability of TreeWise. We also would like to study extension of HMCQ for handling conjunctive queries with built-in predicates. Namely, we are interested in conjunctive queries with homomorphism property such as Left-Semi-Interval(LSI) queries described in [Klu88]. We think concepts of inequality graphs presented in [AM02] could be merged with HMCQ to capture the order of the variables of the query. Also as shown in [AM02], with inequality graph we can capture the possibility of exporting variables, where an existential variable can be treated as distinguished using some appropriate mapping.

Acknowledgements. This work was supported in part by (NSERC) of Canada and by Concordia University. The authors would like to thank Dr. Potinger for making the source code for sample generation available for our experiments.

References

[AD98] Abiteboul, S., Duschka, O.: Complexity of answering queries using materialized views. In: Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Seattle, WA (1998)

[AM02] Afrati, F., Li, C., Prasenjit, M.: Answering queries using views with arithmetic comparisons. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 209–220. ACM Press, New York (2002)

[CM77] Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, pp. 77–90 (1977)

[CS95] Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: Proc. IEEE Int. Conf. on Data Eng., pp. 190–200 (1995)

[Dus97] Duschka, O.M.: Query planning and optimization in information integration. PhD thesis, Stanford University, Stanford, CA (1997)

[Klu88] Klug, A.: On conjunctive queries containing inequalities. Journal of the ACM 35(1), 146–160 (1988)

[Len02] Lenzerini, M.: Data integration: a theoretical perspective. In: PODS 2002: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York, NY, USA, pp. 233–246 (2002)

[Lev01] Levy, A.Y.: Answering queries using views: A survey. The VLDB Journal 10(4), 270–294 (2001)

[LMSS95] Levy, A.Y., Mendelzon, A.O., Sagiv, Y., Srivastava, D.: Answering queries using views. In: Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), San Jose, CA (1995)

[LP90] Levene, M., Poulovassilis, A.: The hypernode model and its associated query language. In: JCIT: Proceedings of the fifth Jerusalem conference on Information technology, pp. 520–530. IEEE Computer Society Press, Los Alamitos (1990)

[LRO96a] Levy, A.Y., Rajaraman, A., Ordille, J.J.: Query answering algorithms for information agents. In: Proceedings of AAAI (1996)

[LRO96b] Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogeneous information sources using source descriptions. In: Proc. Int'l Conf. on Very Large Data Bases (VLDB), Bombay, India (1996)

[PL00] Pottinger, R., Levy, A.Y.: A scalable algorithm for answering queries using views. The VLDB Journal, 484–495 (2000)

[Qia96] Qian, X.: Query folding. In: Proc. IEEE Int'l Conf. on Data Eng. New Orleans, LA (February 1996)

Rewriting Conjunctive Queries over Description Logic Knowledge Bases

Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks

Computing Laboratory
University of Oxford
Oxford, UK
{hector.perez-urbina,boris.motik,ian.horrocks}@comlab.ox.ac.uk

Abstract. We consider the problems of conjunctive query answering and rewriting for information integration systems in which a Description Logic ontology is used to provide a global view of the data. We present a resolution-based query rewriting algorithm for DL-Lite$^+$ ontologies, and use it to show that query answering in this setting is NLogSpace-complete with respect to data complexity. We also show that our algorithm produces an optimal rewriting when the input ontology is expressed in the language DL-Lite. Finally, we sketch an extended version of the algorithm that would, we are confident, be optimal for several DL languages with data complexity of query answering ranging from LogSpace to PTime-complete.

1 Introduction

The use of ontologies as conceptual views over data repositories has proven to be useful in a variety of different scenarios. In Information Integration (II) [15], Enterprise Application Integration (EAI) [14], and the Semantic Web [11], ontologies are used to represent the domain of a given application. This provides users with a coherent global view of the information, thus hiding the details of data organization. In this paper, we focus on II systems in which an ontology is used to provide transparent access to several independent information sources. Typically, such a system consists of a *global* ontology, representing the structure of the application domain, a set of (relational) schemas representing the structure of the *local* information sources, and a set of mappings that relates the global ontology to the local schemas.

The global ontology is often expressed in a Description Logic (DL). DLs are a family of knowledge representation formalisms that model a given domain in terms of concepts (unary predicates), roles (binary predicates), and individuals (constants) [2]. A DL Knowledge Base (KB) consists of a *terminological* component $\mathcal{T}$ called the TBox, and an *assertional* component $\mathcal{A}$ called the ABox. In analogy to databases, the TBox can be seen as a conceptual schema and the ABox as a (partial) instantiation of the schema. The syntax of DLs can be restricted in a variety of ways to trade off the expressive power against computational complexity, and thus to obtain a representation language that is suitable for the application at hand.

K.-D. Schewe and B. Thalheim (Eds.): SDKB 2008, LNCS 4925, pp. 199–214, 2008.

The main task of an II system is to provide a service for answering a query Q over the global ontology using information in the local sources. This service can be realized via query rewriting: the query over the global ontology can be rewritten into a query that is then evaluated over the local sources [10]. Calvanese et al. [5] showed that query rewriting in global-as-view II systems—that is, systems in which concepts and roles from the global ontology are mapped to the local sources by a query over one or more sources [15]—can be solved in two stages. Given a global ontology $\mathcal{T}$ expressed in the description logic DL-Lite [7] and a query Q over $\mathcal{T}$, we can first eliminate $\mathcal{T}$—that is, we can compute a query Q' (which depends on Q and $\mathcal{T}$) such that, for every ABox $\mathcal{A}$, the answers of Q over $\mathcal{T}$ and $\mathcal{A}$, and the answers of Q' over $\mathcal{A}$ coincide; this problem is known as *query rewriting w.r.t. DL TBoxes*. We can then deal with the mappings by unfolding—that is, by replacing each atom in Q' with its definition in the mappings. Assuming mappings are as in [5], this second step is rather straightforward; in contrast, the rewriting of Q and $\mathcal{T}$ into Q' is the main technical problem in the overall algorithm. Therefore, in the rest of this paper, we consider the problem of query rewriting w.r.t. DL TBoxes; the application of our results in an II setting is then straightforward and can be done as in [5].

The rewriting algorithm by Calvanese et al. for the DL-Lite family of languages has been used to show that query answering in DL-Lite is in LogSpace w.r.t. data complexity [7]. Similarly, Rosati used a rewriting algorithm for DL TBoxes expressed in the $\mathcal{EL}$ family of languages [1] to show that query answering in $\mathcal{EL}$ is PTime-complete [18].

In this paper we explore the gap between these two results, and investigate the case where the TBox is expressed in DL-Lite$^+$—a language for which query answering is known to be NLogSpace-hard [7]. We present a query rewriting algorithm for DL-Lite$^+$ and use it to show that query answering in DL-Lite$^+$ can be implemented in NLogSpace, thus closing an open problem from [7]. Moreover, we show that our algorithm exhibits "pay-as-you-go" behavior: it produces an optimal rewriting for TBoxes expressed in a subset of DL-Lite$^+$ for which query answering is in LogSpace. Finally, we provide a sketch showing how this algorithm could be straightforwardly extended to deal with members of the $\mathcal{EL}$ family and beyond. In fact, we are confident that such an algorithm would not only deal with the full spectrum of languages from DL-Lite to $\mathcal{EL}$ extended with inverse roles, universal quantifier, and functionality assertions, but would be optimal with respect to data complexity for all such languages.

2 Preliminaries

2.1 Description Logic DL-Lite$^+$

For A an *atomic concept* and P an *atomic role*, a DL-Lite$^+$ *basic concept* has the form A, $\exists P$, or $\exists P.A$. A *TBox* is a set of *inclusion assertions* of the form $B_1 \sqsubseteq B_2$ or $P_1 \sqsubseteq P_2$, where B_1 and B_2 are basic concepts, and P_1 and P_2 are atomic roles. Without loss of generality, we assume that no TBox contains assertions of the form $\exists P.A \sqsubseteq \exists S.B$: without affecting satisfiability of the TBox,

Table 1. Semantics of DL-Lite$^+$

Semantics of concepts:	Semantics of assertions:	
	$\mathcal{I} \models A(a)$	iff $a^{\mathcal{I}} \in A^{\mathcal{I}}$
$(\exists P)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in P^{\mathcal{I}}\}$	$\mathcal{I} \models P(a, b)$	iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$
$(\exists P.A)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in P^{\mathcal{I}} \wedge y \in A^{\mathcal{I}}\}$	$\mathcal{I} \models B_1 \sqsubseteq B_2$	iff $B_1^{\mathcal{I}} \subseteq B_2^{\mathcal{I}}$
	$\mathcal{I} \models P_1 \sqsubseteq P_2$	iff $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$

each assertion of this form can be replaced with $\exists P.A \sqsubseteq C$ and $C \sqsubseteq \exists S.B$, for C a fresh atomic concept. An *ABox* is a set of *membership assertions* of the form $A(a)$ or $P(a, b)$, where A is an atomic concept, P is an atomic role, and a and b are constants. A DL-Lite$^+$ *knowledge base* (KB) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

An *interpretation* $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty interpretation domain $\triangle^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that maps each concept C to a subset $C^{\mathcal{I}}$ of $\triangle^{\mathcal{I}}$, each role P to a subset $P^{\mathcal{I}}$ of $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$, and each constant a to an element $a^{\mathcal{I}}$ of $\triangle^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ is extended to concepts as shown in the left part of Table 1. An interpretation $\mathcal{I}$ is a model of an inclusion or membership assertion α, written $\mathcal{I} \models \alpha$, if $\mathcal{I}$ and α satisfy the conditions shown in the right part of Table 1. An interpretation $\mathcal{I}$ is a *model* of a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ satisfies each of the inclusion assertions in $\mathcal{T}$ and each of the membership assertions in $\mathcal{A}$. A KB $\mathcal{K}$ is *satisfiable* if it has at least one model; furthermore, $\mathcal{K}$ *logically implies* an assertion α, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

DL-Lite is obtained from DL-Lite$^+$ by disallowing concepts of the form $\exists P.A$ on the left-hand side of inclusion assertions $B_1 \sqsubseteq B_2$. The definition of DL-Lite in [7] additionally allows for inverse roles. Extending DL-Lite$^+$ with inverse roles results in a logic with a PTIME-hard query answering problem [7]. Since our goal in this paper is to investigate NLOGSPACE-complete DLs, we do not consider inverse roles in this paper.

2.2 Conjunctive and Datalog Queries

We use the well-known notions of a first-order signature, terms, variables, and atoms. A *Horn clause* is an expression of the form $H \leftarrow B_1 \wedge \cdots \wedge B_m$, where H is a possibly empty atom and $\{B_i\}$ is a set of atoms. H is called the *head* and the set $\{B_i\}$ is called the *body*. With $\square$ we denote the *empty clause* that has no body atoms and whose head atom is $\bot$. A Horn clause C is *safe* if all variables occurring in the head also occur in the body. A Horn clause is a *fact* if it is safe and does not contain body atoms; instead of $H \leftarrow$, we usually denote such clauses as H. With $\mathsf{var}(C)$ we denote the number of variables in a clause C. The *depth* of a term is defined as $\mathsf{depth}(t) = 0$ for t a constant or a variable and $\mathsf{depth}(f(s_1, \ldots, s_m)) = 1 + \max(depth(s_1), \ldots, depth(s_m))$; the depth of an atom is defined as $\mathsf{depth}(R(t_1, \ldots, t_n)) = \max(\mathsf{depth}(t_1), \ldots, \mathsf{depth}(t_n))$; and the depth of a Horn clause C is $\mathsf{depth}(C) = \max(\mathsf{depth}(H), \mathsf{depth}(B_1), \ldots, \mathsf{depth}(B_m))$.

A *datalog* program P is a set of function-free, safe Horn clauses. The *extensional database (EDB) predicates* of P are those that do not occur in the head atom of any Horn clause in P; all other predicates are called *intensional database (IDB) predicates*. Furthermore, P is *linear* if each Horn clause in P contains at most one IDB predicate in the body.

A *datalog query* Q is a tuple $\langle Q_P, P \rangle$, where Q_P is a *query predicate* and P is a datalog program. A datalog query $Q = \langle Q_P, P \rangle$ is called a *union of conjunctive queries* if Q_P is the only IDB predicate in P and the body of each clause in P does not contain Q_P; furthermore, Q is a *conjunctive query* if it is a union of conjunctive queries and P contains exactly one Horn clause; finally, Q is a *linear datalog query* if P is a linear datalog program. A tuple of constants $\vec{a}$ is an *answer* of a datalog query $Q = \langle Q_P, P \rangle$ on a DL-Lite$^+$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ if and only if $\mathcal{K} \cup P \models Q_P(\vec{a})$, where P is considered to be a set of universally quantified implications with the usual first-order semantics; the set of all answers of Q on $\mathcal{K}$ is denoted by $\mathsf{ans}(Q, \mathcal{K})$.

2.3 Resolution with Free Selection

Resolution with free selection is a well-known calculus that can be used to decide satisfiability of a set of Horn clauses N [4]. The calculus is parameterized by a *selection function* S that assigns to each Horn clause C a nonempty set of atoms such that either $S(C) = \{H\}$ or $S(C) \subseteq \{B_i\}$. The atoms in $S(C)$ are said to be *selected* in C. The resolution calculus with free selection $\mathcal{R}$ consists of the following *resolution inference rule*.

$$\frac{A \leftarrow B_1 \wedge \cdots \wedge B_i \wedge \cdots \wedge B_n \qquad C \leftarrow D_1 \wedge \cdots \wedge D_m}{A\sigma \leftarrow B_1\sigma \wedge \cdots \wedge B_{i-1}\sigma \wedge B_{i+1}\sigma \wedge \cdots \wedge B_n\sigma \wedge D_1\sigma \wedge \cdots \wedge D_m\sigma}$$

As usual, we make a technical assumption that the premises do not have variables in common. The atoms B_i and C must be selected in the corresponding premises by a selection function and $\sigma = \mathsf{MGU}(B_i, C)$; that is, σ is the *most general unifier* of B_i and C as defined in [3]. The two clauses above the inference line are called the *premises* and the clause below the line is called the *resolvent*.

An *inference* is an application of an inference rule to concrete premises. A set of Horn clauses N is *saturated* by $\mathcal{R}$ if, for any two premises $P_1, P_2 \in N$, the set N contains a clause that is equivalent to the resolvent of P_1 and P_2 up to variable renaming. A *derivation* by $\mathcal{R}$ from a set of Horn clauses N is a sequence of sets of Horn clauses $N = N_0, N_1, \ldots$ such that, for each $i \geq 0$, we have that $N_{i+1} = N_i \cup \{C\}$, where C is the conclusion of an inference by $\mathcal{R}$ from premises in N_i. A derivation is said to be *fair* if, for any i and any two Horn clauses $P_1, P_2 \in N_i$ to which resolution is applicable, some $j \geq i$ exists such that $R \in N_j$, where R is the resolvent between P_1, P_2. The limit N_∞ of a (possibly infinite) fair derivation from a set of Horn clauses N is defined as $N_\infty = \bigcup_i N_i$. It is well known that N_∞ is saturated by $\mathcal{R}$ and does not depend on the derivation [4]. A set of Horn clauses N is satisfiable if and only if $\square \notin N_\infty$. A clause C is said to be *derivable* from N iff $C \in N_\infty$.

3 Answering Conjunctive Queries in DL-Lite$^+$

Given a DL-Lite$^+$ TBox $\mathcal{T}$ and a conjunctive query $Q = \langle Q_P, \{Q_C\}\rangle$, our goal is to compute a *rewriting* $\mathsf{rew}(Q, \mathcal{T})$—that is, a query such that, for each ABox $\mathcal{A}$, evaluating $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ and evaluating the query Q directly over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ produces exactly the same answers. We derive this algorithm in two steps. In this section, we first show how to compute the set of answers $\mathsf{ans}(Q, \mathcal{K})$ directly; then, in Section 4 we use this result to derive the rewriting algorithm.

It is well known that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable, where $\Xi(\mathcal{K})$ is the set of clauses obtained by transforming $\mathcal{K}$ as shown in Table 2. Therefore, to answer Q over $\mathcal{K}$, we need a decision procedure for checking satisfiability of the latter set of clauses. We derive this procedure using the principles outlined by Joyner [12].

Given $\mathcal{K}$ and Q, with $\mathcal{N}$ we denote the set of clauses of the forms shown in Table 2 that can be constructed using the signature of $\mathcal{K}$ and Q. It is not difficult to see that $\mathcal{N}$ is finite assuming that $\mathcal{K}$ and Q are finite. Furthermore, clearly, if we translate $\mathcal{K}$ into a set of clauses $\Xi(\mathcal{K})$, then $\Xi(\mathcal{K}) \cup \{Q_C\} \subseteq \mathcal{N}$. Finally, we saturate $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ using $\mathcal{R}^{DL}$—a suitably parameterized resolution with free selection calculus. Since $\mathcal{R}^{DL}$ is sound and complete, in order to obtain a decision procedure we only need to show that each saturation terminates. This is done in the key Lemma 2, which shows that the resolvent of any two premises in $\mathcal{N}$ by $\mathcal{R}^{DL}$ is also a clause in $\mathcal{N}$. This immediately implies termination: in the worst case, the saturation derives the entire set $\mathcal{N}$, which is finite.

We now formalize our calculus. We first define the set of clauses $\mathcal{N}$ and parameterize suitably the calculus of resolution with free selection.

Definition 1. *Let $\mathcal{K}$ be a DL-Lite$^+$ knowledge base and $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query. The set of clauses $\Xi(\mathcal{K})$ is obtained by transforming $\mathcal{K}$ as shown in Table 2. Furthermore, the set of DL-Lite$^+$ clauses $\mathcal{N}$ is the set of all clauses of types shown in Table 2 constructed using the symbols in Q_C and $\Xi(\mathcal{K})$.*

With $\mathcal{R}^{DL}$ we denote the resolution calculus with free selection parameterized with the following selection function S.

- *In a clause C of type A1–K2, the selection function S selects the atoms that are underlined in Table 2.*
- *In a clause C of type Q1, the selection function S selects the head atom if C contains functional terms in the head or if the body of C is empty; otherwise, S selects all deepest body atoms of C.*

For N a set of clauses, N_∞ is the limit of a fair derivation from N by $\mathcal{R}^{DL}$.

The principles outlined before Definition 1 allow us only to check whether some tuple $\vec{a}$ is an answer to Q over $\mathcal{K}$. Often, however, we need to compute the entire set $\mathsf{ans}(Q, \mathcal{K})$. This can be done using the *answer literal* technique [9]: instead of saturating $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$, we can saturate just $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$. The following lemma shows that by doing so we shall compute all answers to Q over $\mathcal{K}$.

Lemma 1. *Let $\mathcal{K}$ be a DL-Lite$^+$ knowledge base, $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query, and $\vec{a}$ a tuple of constants. Then, we have $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$.*

Proof. Clearly, $\mathcal{K} \cup \{Q_C\} \models Q_P(\vec{a})$ iff $\mathcal{K} \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable; we now prove that the latter is the case iff $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$. The ($\Leftarrow$) direction is trivial. For the ($\Rightarrow$) direction, note that $\Xi(\mathcal{K}) \cup \{Q_C\}$ does not contain a clause with the empty head, so a saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$ cannot derive the empty clause. Furthermore, the predicate Q_P does not occur in the body of any clause in $\Xi(\mathcal{K}) \cup \{Q_C\}$; hence, $\mathcal{R}^{DL}$ can derive the empty clause from $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$. □

Thus, to compute $\mathsf{ans}(Q, \mathcal{K})$, we simply need to saturate $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$. The following key lemma shows that such a saturation terminates.

Lemma 2. *For each two clauses $C_1, C_2 \in \mathcal{N}$ and C_r the resolvent of C_1 and C_2 by $\mathcal{R}^{DL}$, we have that $C_r \in \mathcal{N}$.*

Proof. Let C_1 and C_2 be some clauses of $\mathcal{N}$, and let C_r be a resolvent of C_1 and C_2 by $\mathcal{R}^{DL}$. The possible inferences by $\mathcal{R}^{DL}$ on C_1 and C_2 are summarized in Table 3. As can be seen from the table, if C_1 and C_2 are of types A1–K2, then C_r is also of type A1–K2.

Assume that C_1 is of type Q1, satisfying properties (*i*)–(*iii*) of Table 2. If the head atom $Q_P(\vec{t})$ of C_1 is selected, then resolution is not possible, since no clause in $\mathcal{N}$ contains Q_P in the body. If a unary body atom $A(t)$ of C_1 is selected, then

Table 2. Clause Set $\mathcal{N}$ for $Q = \langle Q_P, \{Q_C\}\rangle$ and $\mathcal{K}$

Type	DL-Lite$^+$ clause	DL-Lite$^+$ axiom
A1	$A(a)$	$A(a)$
A2	$P(a, b)$	$P(a, b)$
T1	$B(x) \leftarrow A(x)$	$A \sqsubseteq B$
T2	$P(x, f^i_A(x)) \leftarrow A(x)$	$A \sqsubseteq \exists P.B$
T3	$B(f^i_A(x)) \leftarrow A(x)$	
T4	$B(x) \leftarrow P(x, y)$	$\exists P \sqsubseteq B$
T5	$B(x) \leftarrow P(x, y) \wedge A(y)$	$\exists P.A \sqsubseteq B$
T6	$S(x, y) \leftarrow P(x, y)$	$P \sqsubseteq S$
K1	$B(a) \leftarrow A(b)$	
K2	$B_2(x) \leftarrow B_1(f^i_A(x)) \wedge A(x)$	
Q1	$Q_P(\vec{t}) \leftarrow \bigwedge L_i(\vec{t_i})$	

Note 1. We use A and B to denote atomic concepts, P and S for atomic roles, L_i for atomic concepts or roles, and $\vec{t}$ (possibly subscripted) for a tuple of terms. Each axiom of the form $A \sqsubseteq \exists P.B$ is uniquely associated with a function symbol f^i_A. For each clause C of type Q1, (*i*) $\mathsf{var}(C) \leq \mathsf{var}(Q_C)$, (*ii*) $\mathsf{depth}(C) \leq \max(1, \mathsf{var}(Q_C) - \mathsf{var}(C))$, and (*iii*) if a variable x occurs in a functional term in C, then x occurs in all functional terms in C.

Table 3. Inferences of $\mathcal{R}^{DL}$ on $\mathcal{N}$

A1 + T1 = A1:

$$\frac{\underline{A(a)} \quad B(x) \leftarrow \underline{A(x)}}{B(a)}$$

T2 + T4 = T1:

$$\frac{\underline{P(x, f^i_A(x))} \leftarrow A(x) \quad B(x) \leftarrow \underline{P(x,y)}}{B(x) \leftarrow A(x)}$$

A2 + T4 = A1:

$$\frac{\underline{P(a,b)} \quad B(x) \leftarrow \underline{P(x,y)}}{B(a)}$$

T2 + T5 = K2:

$$\frac{\underline{P(x, f^i_A(x))} \leftarrow A(x) \quad B(x) \leftarrow \underline{P(x,y)} \wedge C(y)}{B(x) \leftarrow C(f^i_A(x)) \wedge A(x)}$$

A2 + T5 = K1:

$$\frac{\underline{P(a,b)} \quad B(x) \leftarrow \underline{P(x,y)} \wedge A(y)}{B(a) \leftarrow A(b)}$$

T2 + T6 = T2:

$$\frac{\underline{P(x, f^i_A(x))} \leftarrow A(x) \quad S(x,y) \leftarrow \underline{P(x,y)}}{S(x, f^i_A(x)) \leftarrow A(x)}$$

A2 + T6 = A2:

$$\frac{\underline{P(a,b)} \quad S(x,y) \leftarrow \underline{P(x,y)}}{S(a,b)}$$

K1 + A1 = A1:

$$\frac{\underline{B(a)} \leftarrow \underline{A(b)} \quad \underline{A(b)}}{B(a)}$$

T1 + T3 = T3:

$$\frac{\underline{C(x)} \leftarrow \underline{B(x)} \quad \underline{B(f^i_A(x))} \leftarrow A(x)}{C(f^i_A(x)) \leftarrow A(x)}$$

K2 + T3 = T1:

$$\frac{\underline{C(x)} \leftarrow \underline{B(f^i_A(x))} \wedge A(x) \quad \underline{B(f^i_A(x))} \leftarrow A(x)}{C(x) \leftarrow A(x)}$$

Q1 + A1 = Q1:

$$\frac{\underline{Q_P(\vec{u})} \leftarrow \underline{A(t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{A(a)}}{Q_P(\vec{u})\sigma \leftarrow \bigwedge L_i(\vec{t_i})\sigma}$$

Q1 + A2 = Q1:

$$\frac{\underline{Q_P(\vec{u})} \leftarrow \underline{P(s,t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{P(a,b)}}{Q_P(\vec{u})\sigma \leftarrow \bigwedge L_i(\vec{t_i})\sigma}$$

Q1 + T3 = Q1:

$$\frac{\underline{Q_P(\vec{u})} \leftarrow \underline{A(t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{A(f^i_B(x))} \leftarrow B(x)}{Q_P(\vec{u})\sigma \leftarrow B(x)\sigma \wedge \bigwedge L_i(\vec{t_i})\sigma}$$

Q1 + T2 = Q1:

$$\frac{\underline{Q_P(\vec{u})} \leftarrow \underline{P(s,t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{P(x, f^i_A(x))} \leftarrow A(x)}{Q_P(\vec{u})\sigma \leftarrow A(x)\sigma \wedge \bigwedge L_i(\vec{t_i})\sigma}$$

Note 2. The notation $A + B = C$ denotes that "resolving a clause of type A with a clause of type B produces a clause of type C."

C_2 can be of type A1 or T3; we now show that C_r satisfies properties (i)–(iii) of Table 2.

- If C_2 is of type A1, unification is possible only if the term t is either a constant a or a variable y. In the former case, the unifier σ is empty; in the latter case, $\sigma = \{y \mapsto a\}$. Clearly, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$ and $\mathsf{depth}(C_r) = \mathsf{depth}(C_1)$, so C_r satisfies (i) and (ii). Furthermore, since $A(t)$ is the deepest atom in C_1, the

clause C_1 does not contain functional terms, so C_r does not contain them either; hence, C_r satisfies (*iii*) vacuously.

– If C_2 is of type T3, unification is possible only if the term t is a variable or a functional term.
  - If t is a variable y, then $\sigma = \{y \mapsto f^i_B(x)\}$. Clearly, $\mathsf{var}(C_r) = \mathsf{var}(C_1)$, so C_r satisfies (*i*). Furthermore, $\mathsf{depth}(C_1) = 0$ and $\mathsf{depth}(C_r) \leq 1$, so C_r satisfies (*ii*). Finally, every occurrence of y is replaced with $f^i_B(x)$, and C_1 does not contain functional terms, so (*iii*) holds as well.
  - If t is a functional term $f^i_B(s)$, the unifier is of the form $\sigma = \{x \mapsto s\}$. Clearly, $\mathsf{var}(C_r) = \mathsf{var}(C_1)$, so C_r satisfies (*i*). Furthermore, since no term in C_1 is deeper than $f^i_B(s)$, we have $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1)$, so C_r satisfies (*ii*). Finally, the inference does not introduce new functional terms, so C_r satisfies (*iii*).

If a binary atom $P(s,t)$ is selected in C_1, then C_2 can be of type A2 or T2. We now show that C_r satisfies properties (*i*)–(*iii*) of Table 2.

– If C_2 is of type A2, the unification is possible only if the terms s and t are not functional terms. If they are both constants, the substitution σ is empty; otherwise, σ maps s, t, or both to the corresponding constants in C_2. Clearly, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$, so C_r satisfies (*i*). Furthermore, $\mathsf{depth}(C_r) = \mathsf{depth}(C_1)$, so C_r satisfies (*ii*). Finally, since $P(s,t)$ is the deepest atom in C_1, the clause C_1 does not contain functional terms, so C_r satisfies (*iii*) vacuously.
– If C_2 is of type T2, unification is possible only if the term t is a variable or a functional term.
  - If t is a variable x_t, then $\sigma = \{x_t \mapsto f^i_A(s), x \mapsto s\}$. Due to the occurs-check in unification, x_t cannot occur in s. The inference thus decreases the number of variables of C_1 in C_r by one: $\mathsf{var}(C_r) = \mathsf{var}(C_1) - 1$, so C_r satisfies (*i*). Furthermore, C_1 satisfies (*iii*), so x_t does not occur in a functional term in C_1 (because it does not occur in s). Hence, even though x_t is mapped to a functional term, $\mathsf{depth}(C_r) = \mathsf{depth}(C_1) + 1$, so C_r satisfies (*ii*). Finally, since every occurrence of x_t is replaced with $f^i_A(s)$, C_r satisfies (*iii*) as well.
  - Assume that t is a functional term $f^i_A(t')$. If s does not occur in t', then s is a variable x_s and $\sigma = \{x \mapsto t', x_s \mapsto t'\}$. If s occurs in t', the only way for the inference to be possible is if $t' = s$, so $\sigma = \{x \mapsto t'\}$. In both cases, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$ and $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1)$, so C_r satisfies properties (*i*) and (*ii*). Furthermore, the inference does not introduce new functional terms, so C_r satisfies (*iii*) as well.

This covers all possible forms of C_1 and C_2, so the lemma holds. □

This lemma now straightforwardly implies that a saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ terminates, so we can use it to compute $\mathsf{ans}(Q, \mathcal{K})$.

Lemma 3. *For $\mathcal{K}$ a DL-Lite$^+$ knowledge base and $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query, the saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$ terminates.*

Proof. The clause set $\mathcal{N}$ for Q and $\mathcal{K}$ is finite. Moreover, no clause is ever deleted during the saturation process and, by Lemma 2, $\mathcal{N}$ is closed under $\mathcal{R}^{DL}$. Hence, in the worst case, $\mathcal{R}^{DL}$ derives all clauses in $\mathcal{N}$ and then terminates. □

We believe that the result in this section can relatively easily be extended to more expressive DLs, such as $\mathcal{EL}$ extended with inverse roles, universal quantifiers in the implication consequent, and functionality assertions. The key to achieving this is to extend the clause set in Table 2 to cover the new constructors, and to prove that Lemma 2 still holds. We believe we can do this along the lines of [17,16]; the main technical difficulty is to precisely describe the interaction between the new types of clause and clauses of type Q1. Once this is done, however, the rest of this paper (i.e., the material in the following two sections), should hold with only minor changes. The resulting algorithm would deal with the full spectrum of languages from DL-Lite to extended $\mathcal{EL}$, and we are confident that it would still be optimal with respect to data complexity for all such languages.

4 Rewriting Conjunctive Queries in DL-Lite$^+$

The algorithm from the previous section allows us to compute the answers to a conjunctive query $Q = \langle Q_P, \{Q_C\}\rangle$ over a DL-Lite$^+$ knowledge base $\mathcal{K}$. If we ask the same query over different ABoxes, the algorithm will repeat a lot of unnecessary work, since the query answering algorithm depends on both $\mathcal{T}$ and $\mathcal{A}$. In this section, we present an algorithm for *query rewriting*: given Q and a TBox $\mathcal{T}$, we compute a datalog query $\mathsf{rew}(Q, \mathcal{T})$ such that, for any ABox $\mathcal{A}$, the sets of answers of Q over $\langle \mathcal{T}, \mathcal{A}\rangle$ and of $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ are the same. Thus, our algorithm eliminates the TBox and reduces the problem of answering conjunctive queries in DL-Lite$^+$ to the problem of answering datalog queries.

A distinguishing feature of our algorithm is that it exhibits "pay-as-you-go" behavior: the resulting datalog program $\mathsf{rew}(Q, \mathcal{T})$ is optimal for the input TBox $\mathcal{T}$. If $\mathcal{T}$ is in DL-Lite$^+$, then $\mathsf{rew}(Q, \mathcal{T})$ consists of a union of conjunctive queries and a linear datalog program. We use this fact in Section 5 to establish novel data complexity bounds for conjunctive query answering. Furthermore, if $\mathcal{T}$ is in DL-Lite, then $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries. Hence, our algorithm generalizes the approach from [7].

We derive the rewriting algorithm in two phases: in Section 4.1, we show how to convert $\Xi(\mathcal{T})$ into a nonoptimal datalog program by eliminating function symbols; then, in Section 4.2 we present an additional step that ensures that the rewritten query is of optimal form.

4.1 Elimination of Function Symbols

The following definition summarizes the first step of our rewriting algorithm.

Definition 2. *For $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query and $\mathcal{T}$ a DL-Lite$^+$ TBox, $\mathsf{ff}(Q, \mathcal{T})$ is the set that contains exactly all function-free clauses contained in $(\Xi(\mathcal{T}) \cup \{Q_C\})_\infty$.*

We next show that, for each ABox $\mathcal{A}$, we have $\{Q_C\} \cup \mathcal{T} \cup \mathcal{A} \models Q_P(\vec{a})$ if and only if $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$. Thus, $\mathsf{ff}(Q, \mathcal{T})$ is a rewriting of Q and $\mathcal{T}$, albeit not necessarily an optimal one. We prove the claim proof-theoretically: we show that $Q_P(\vec{a})$ is derivable from $\{Q_C\} \cup \mathcal{T} \cup \mathcal{A}$ if and only if it is derivable from $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A}$. To this end, we first prove that we can always "postpone" the inferences with the ABox clauses in the saturation of $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$—that is, we can first perform all inferences with nonground clauses only, and then perform the inferences involving a ground clause.

Lemma 4. *Let $Q = \langle Q_P, \{Q_C\}\rangle$ be a conjunctive query, $\mathcal{T}$ a DL-Lite$^+$ TBox, and $\mathcal{A}$ an ABox. For each clause C of type Q1 derivable from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$, a clause C' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ such that, for G the subset of all clauses of type A1 and A2 in $(\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A})_\infty$, we have $\{C'\} \cup G \models C$.*

Proof. We prove the claim by induction on the height of a derivation tree by which C is derived from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$. If the derivation tree has height zero, then C must be the clause Q_C, so the claim follows trivially for $C' = Q_C$. Assume that the claim holds for each clause derived from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$ by a derivation tree of height n, and consider a clause C derived by a derivation tree of height $n + 1$. The clause C is obtained by resolving some clauses C_1 and C_2. According to Table 3, one of the premises has to be of type Q1, so we denote it by C_1; the other premise C_2 can be of type A1, A2, T2, or T3. By the induction hypothesis, some clause C_1' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ such that $\{C_1'\} \cup G \models C_1$. We now consider the different forms that C_2 can have.

Assume that C_2 is of type A1 or A2. From Table 3 we can see that each derivation of a clause of type A1 or A2 involves only function-free clauses, so $C_2 \in G$. The inductive claim now trivially holds for $C' = C_1'$.

Assume that C_2 is of type T2 or T3. We assume that it is of the form $L(\vec{t}) \leftarrow A(x)$; then, C_1 must contain in the body a counterpart atom $L(\vec{q})$. By examining the inferences between DL-Lite$^+$ clauses shown in Table 3, we can see that C_2 is derivable from $\Xi(\mathcal{T})$. Note that G contains only ground clauses of types A1 and A2; thus, since $\{C_1'\} \cup G \models C_1$, a subset $\{G_i(\vec{a_i})\} \subseteq G$ exists such that resolving C_1' on body literals $\{G_i(\vec{g_i})\}$ with the elements of $\{G_i(\vec{a_i})\}$ produces C_1. Furthermore, all such resolution inferences just remove body atoms. Therefore, if C_1 is to contain the atom $L(\vec{q})$ in the body, the clause C_1' must contain an atom $L(\vec{s})$ in its body. Hence, C_1' is of the form (1), and C_1 is of the form (2), where δ maps some variables to constants in $\{G_i(\vec{a_i})\}$ such that $L(\vec{s})\delta = L(\vec{q})$. Finally, resolving C_1 and C_2 produces the clause C, which is of the form (3) for $\sigma = \mathsf{MGU}(L(\vec{s})\delta, L(\vec{t}))$.

$$C_1' = Q_P(\vec{u}) \leftarrow L(\vec{s}) \wedge \bigwedge G_i(\vec{g_i}) \wedge \bigwedge M_j(\vec{m_j}) \tag{1}$$

$$C_1 = Q_P(\vec{u})\delta \leftarrow L(\vec{s})\delta \wedge \bigwedge M_j(\vec{m_j})\delta \tag{2}$$

$$C = Q_P(\vec{u})\delta\sigma \leftarrow A(x)\sigma \wedge \bigwedge M_j(\vec{m_j})\delta\sigma \tag{3}$$

Note that no inference used to derive C_1 changes the number of function symbols of C_1'; therefore, $L(\vec{s})$ is the deepest literal of C_1'. Furthermore, each variable of

C_1' that is replaced by δ with a constant clearly does not occur in $L(\vec{s})\delta$; hence, the substitutions δ and σ have disjoint domains, and $\sigma = \delta\sigma$.

We now transform this derivation into a derivation in which all inferences with ABox clauses are performed after all inferences with only TBox clauses. Let C' be the clause obtained by resolving C_1' and C_2; we can assume that C' has the form (4), where $\sigma' = \mathsf{MGU}(L(\vec{s}), L(\vec{t}))$.

$$C' = Q_P(\vec{u})\sigma' \leftarrow A(x)\sigma' \wedge \bigwedge G_i(\vec{g_i})\sigma' \wedge \bigwedge M_j(\vec{m_j})\sigma' \tag{4}$$

Since $L(\vec{s})$ is the deepest literal of C_1', the inference between C_1' and C_2 satisfies the selection function of the calculus $\mathcal{R}^{DL}$. Since both C_1' and C_2 are derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$, the clause C' is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ as well. Let x be a variable that occurs in $L(\vec{s})$ and that is replaced by δ with a constant. Clearly, σ does not contain such x; hence, without loss of generality, we can assume (*) that σ' does not contain such variables either—that is, instead of mapping x to a term in $L(\vec{t})$, we can assume that the corresponding term is mapped to x.

Let D now be the clause obtained by resolving the literals $G_i(\vec{g_i})\sigma'$ in C' with the ground clauses $\{G_i(\vec{a_i})\}$. This inference is possible due to assumption (*), so D has the following form, where δ' maps some variables of C' to constants.

$$D = Q_P(\vec{u})\sigma'\delta' \leftarrow A(x)\sigma'\delta' \wedge \bigwedge M_j(\vec{m_j})\sigma'\delta' \tag{5}$$

Due to (*), σ and σ' have the same domain which is disjoint with the domain of δ, so $\sigma = \sigma'\delta$. None of the variables occurring in $\{G_i(\vec{g_i})\}$ is in the domain of σ', so $\delta = \delta'$. Since $\sigma = \sigma'\delta$ and $\delta = \delta'$, we have $\sigma = \sigma'\delta'$. Moreover, since $\sigma = \delta\sigma$, we have $\sigma = \delta\sigma = \sigma'\delta'$, so $C = D$, which proves our claim. □

This lemma now allows us to prove the desired relationship between the answers of Q on $\mathcal{T}$ and $\mathcal{A}$ and the answers of $\mathsf{ff}(Q, \mathcal{T})$ on $\mathcal{A}$.

Lemma 5. *Let $Q = \langle Q_P, \{Q_C\}\rangle$ be a conjunctive query, $\mathcal{T}$ a DL-Lite$^+$ TBox, and $\mathcal{A}$ an ABox. Then, $\vec{a} \in \mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A}\rangle)$ if and only if $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$.*

Proof. ($\Leftarrow$) Note that $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \subseteq (\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A})_\infty$, which trivially implies this direction of the claim.

($\Rightarrow$) Assume that $Q_P(\vec{a})$ is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$. Since $Q_P(\vec{a})$ is of type Q1, by Lemma 4, a clause C' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ such that, for G the subset of all clauses of type A1 and A2 in $(\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A})_\infty$, we have $\{C'\} \cup G \models C$. Since $Q_P(\vec{a})$ does not contain function symbols, C' cannot contain function symbols either, so $C' \in \mathsf{ff}(Q, \mathcal{T})$. Thus, $Q_P(\vec{a})$ is implied by $\mathsf{ff}(Q, \mathcal{T}) \cup G$ so, by the definition of G, we have $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$. □

4.2 Optimizing the Rewriting through Unfolding

By Lemma 5, the datalog program $\mathsf{ff}(Q, \mathcal{T})$ is a rewriting of Q w.r.t. $\mathcal{T}$; however, it is not necessarily optimal for the TBox $\mathcal{T}$ at hand. In particular, the program

$\mathsf{ff}(Q, \mathcal{T})$ can contain clauses of type T1, T4, and T6; hence, we must assume that each predicate in $\mathsf{ff}(Q, \mathcal{T})$ can be an IDB predicate, so a clause of type T4 is not a linear datalog rule. Our goal, however, is to ensure that the rewriting consists of a linear datalog program and a union of conjunctive queries; furthermore, if $\mathcal{T}$ is a DL-Lite TBox, the rewriting should be a union of conjunctive queries only. Thus, in this section we introduce a further *unfolding* step that transforms $\mathsf{ff}(Q, \mathcal{T})$ into a datalog program of an optimal form.

Definition 3. *The* unfolding *of* $L(\vec{x}) \leftarrow \bigwedge M_i(\vec{m_i})$ *in* $N(\vec{n}) \leftarrow L(\vec{x'}) \wedge \bigwedge P_j(\vec{p_j})$ *is the clause* $N(\vec{n})\sigma \leftarrow \bigwedge M_i(\vec{m_i})\sigma \wedge \bigwedge P_j(\vec{p_j})\sigma$*, where* $\sigma = \mathsf{MGU}(L(\vec{x}), L(\vec{x'}))$.

Given two sets of safe Horn clauses R *and* U*, let* R_U *be the smallest set such that* $R \subseteq R_U$ *and, for each unfolding* C_r *of a clause* $C_1 \in R \cap U$ *in a clause* $C_2 \in R$*, we have that* $C_r \in R_U$*. The* unfolding *of* R *w.r.t.* U *is defined as* $\mathsf{unfold}(R, U) = R_U \setminus U$.

We shall apply the unfolding step for $R = \mathsf{ff}(Q, \mathcal{T})$ and U the set of all clauses of types T1, T4, and T6. Since unfolding eliminates all clauses of type T6, all atomic roles thus become EDB predicates; thus, the resulting set of clauses, apart from the clauses of type Q1, is a linear datalog program. Moreover, since all clauses of types T1 and T4 are also eliminated, the resulting set of clauses becomes a union of conjunctive queries whenever $\mathcal{T}$ is a DL-Lite TBox. Before proceeding, however, we show that unfolding does not change the set of "relevant" consequences of a datalog program.

Lemma 6. *Let* R *and* U *be sets of safe Horn clauses. For any set of facts* A *and for any predicate* F *that does not occur in* U*, we have* $R \cup A \models F(\vec{a})$ *if and only if* $\mathsf{unfold}(R, U) \cup A \models F(\vec{a})$.

Proof. ($\Leftarrow$) Note that $R \models R_U$ and $\mathsf{unfold}(R, U) \subseteq R_U$; therefore, for each clause C, if $\mathsf{unfold}(R, U) \cup A \models C$ then $R \cup A \models C$.

($\Rightarrow$) Let $\mathcal{H}$ be the hyperresolution calculus—that is, the resolution calculus in which all the body literals are selected whenever possible, and in which all selected literals are resolved in one step. It is well known that, if $R \models F(\vec{a})$, then a derivation tree T for $F(\vec{a})$ from R by $\mathcal{H}$ exists [4]. We represent such a tree T as a tuple $\langle T_N, \delta, \lambda \rangle$ for T_N the set of nodes where we denote with $t.i$ the i-th child of $t \in T_N$ and with ϵ the root node; δ is a function that maps each node $t \in T_N$ to a fact $\delta(t)$; and λ is a function that maps each node $t \in T_N$ to a clause $\lambda(t)$ such that $\delta(t)$ is derived by hyperresolving each $\delta(t.i)$ with the i-th body literal of $\lambda(t)$. If t is a leaf node, then $\delta(t) \in A$ and $\lambda(t)$ is undefined.

We now inductively define a function $\sigma(t)$ as follows: starting from the leaves upwards, for each $t \in T_N$, we set $\sigma(t)$ to be the clause obtained from $\lambda(t)$ by unfolding each $\sigma(t.i)$ in the i-th body atom of $\lambda(t)$ provided that $\sigma(t.i) \notin U$ or $\delta(t.i) \in A$; furthermore, we call t a *surviving* node iff $\sigma(t) \notin U$ or $\delta(t) \in A$. We say that a node t_2 is the *closest surviving node* to t_1 if t_2 is a surviving node, if it is a descendent of t_1, and no node on the path between t_1 and t_2 is a surviving node. By the inductive definition of σ, it is easy to see that, for each node t, the fact $\delta(t)$ can be derived by hyperresolving $\sigma(t)$ with the set

of facts $\{\delta(t_1), \ldots, \delta(t_n)\}$, where $t_1, \ldots, t_n$ are exactly all the closest surviving nodes to t. Note that, for every node $t \in T_N$, we have $\sigma(t) \in R_U$. Moreover, if t is a surviving node, then $\sigma(t) \in \mathsf{unfold}(R, U)$. Therefore, if t is a surviving node, the fact $\delta(t)$ can be derived from $\mathsf{unfold}(R, U) \cup A$.

Since the predicate F does not occur in U, we have $F(\vec{a}) \notin U$. Furthermore, $\delta(\epsilon) = F(\vec{a})$, so the clause $\sigma(\epsilon)$ contains F in the head, and $\sigma(\epsilon) \notin U$. Thus, ϵ is a surviving node, so $\delta(\epsilon)$ can be derived from $\mathsf{unfold}(R, U) \cup A$. □

We are now ready to define the rewriting of a conjunctive query Q with respect to a TBox $\mathcal{T}$ expressed in DL-Lite$^+$.

Definition 4. *The* rewriting $\mathsf{rew}(Q, \mathcal{T})$ *of a conjunctive query* $Q = \langle Q_P, \{Q_C\}\rangle$ *w.r.t. a DL-Lite*$^+$ *TBox* $\mathcal{T}$ *is the query* $\langle Q_P, \mathsf{unfold}(R, U)\rangle$*, where* $R = \mathsf{ff}(Q, \mathcal{T})$ *and* U *is the subset of* $\mathcal{N}$ *of all clauses of type T1, T4, and T6.*

We now state the main property of the reduction algorithm.

Theorem 1. *For a conjunctive query* Q*, a DL-Lite*$^+$ *TBox* $\mathcal{T}$*, and an ABox* $\mathcal{A}$*, we have* $\mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A}\rangle) = \mathsf{ans}(\mathsf{rew}(Q, \mathcal{T}), \mathcal{A})$.

Proof. Without loss of generality, we can assume that Q_P does not occur in $\Xi(\mathcal{T})$. Then, the claim of this theorem follows straightforwardly from Lemmata 5 and 6. □

We now prove two important properties about the structure of the rewriting. We use these properties in Section 5 to prove complexity results about answering conjunctive queries over DL-Lite$^+$.

Lemma 7. *Let* $Q = \langle Q_P, \{Q_C\}\rangle$ *be a conjunctive query,* $\mathcal{T}$ *a DL-Lite*$^+$ *TBox, and* $\mathsf{rew}(Q, \mathcal{T}) = \langle Q_P, P\rangle$*. Then,* P *can be split into disjoint subsets* U_Q *and* U_C *such that* $\langle Q_P, U_Q\rangle$ *is a union of conjunctive queries and* $\langle Q_P, U_C\rangle$ *is a linear datalog query.*

Proof. Let $U_Q \subseteq P$ be the set of all clauses of type Q1 in P. By the definition of clauses of type Q1, Q_P is the head predicate of every clause in U_Q, and Q_P does not appear in the body of a clause in U_Q, so $\langle Q_P, U_Q\rangle$ is a union of conjunctive queries. Let $U_C = P \setminus U_Q$. The program $\mathsf{ff}(Q, \mathcal{T})$ contains clauses of types T1, T4, T5, and T6. Hence, U_C is obtained by unfolding clauses of types T1, T4, and T6 in clauses of types T1, T4, T5, and T6, and then by removing all clauses of types T1, T4, and T6. Thus, U_C contains clauses of type T5 and clauses of the form $B(x) \leftarrow P(x, y) \wedge S(y, z)$ that are obtained by unfolding a clause of type T4 in a clause of type T5. Clearly, no clause in U_C contains a role predicate in the head, so all role predicates are EDB predicates. Furthermore, clauses of type T5 can contain a unary predicate in the head, so unary predicates can be IDB predicates; however, IDB predicates can occur only in a clause of type T5 in the body, so all such clauses are linear. Thus, U_C is a linear datalog program. □

Lemma 8. *For* $Q = \langle Q_P, \{Q_C\}\rangle$ *a conjunctive query and* $\mathcal{T}$ *a DL-Lite TBox,* $\mathsf{rew}(Q, \mathcal{T})$ *is a union of conjunctive queries.*

Proof. Let $\mathsf{rew}(Q, \mathcal{T}) = \langle Q_P, P \rangle$. Since $\mathcal{T}$ is a DL-Lite TBox, the set $\Xi(\mathcal{T})$ does not contain clauses of type T5. Thus, $\mathsf{ff}(Q, \mathcal{T})$ contains only clauses of types Q1, T1, T4, and T6. Clauses of types T1, T4, and T6 are unfolded in clauses of type Q1, so $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries. □

5 Complexity Analysis

It is well known that the problem of deciding $P \models A(\vec{a})$ for P a linear datalog program is NLogSpace-complete with respect to data complexity [8]. We were not able to find in the literature a generalization of this result for the case where P consists of a linear datalog program and a union of conjunctive queries; therefore, before proceeding, we show that this is indeed the case.

Lemma 9. *For $Q = \langle Q_P, Q_C \rangle$ a union of conjunctive queries, P a linear datalog program, and A a set of facts, deciding $P \cup Q_C \cup A \models Q_P(\vec{a})$ can be performed in* NLogSpace *in the size of A.*

Proof. If $P \cup Q_C \cup A \models Q_P(\vec{a})$, then $Q_P(\vec{a})$ can be derived from the set of clauses $P \cup Q_C \cup A$ using SLD resolution [4]. First, we nondeterministically choose a query $Q_i \in Q_C$ and ground it by nondeterministically choosing a set of constants from A. We then initialize the *goal* G to be the resolvent of Q_i and $\leftarrow Q_P(\vec{a})$; if resolution is not possible, the algorithm halts. Then, we start the following loop. We first eliminate all atoms with EDB predicates in G by resolving them with facts in A; if some atom cannot be resolved, the algorithm halts. If G has an empty body, the algorithm accepts. Otherwise, we nondeterministically choose a rule $R \in P$ and generate its grounding R' by nondeterministically choosing a set of constants from A. Finally, we set our goal G to be the resolvent between R' and G; if this is not possible, the algorithm halts. We now repeat the loop. To ensure termination, we maintain a counter that is initialized in the beginning to the number of ground clauses of P and A multiplied by the number of the query rules in Q_C. We decrease the counter after each pass through the loop, and we terminate the loop if the counter reaches zero. Clearly, if the algorithm accepts, then SLD resolution for $Q_P(\vec{a})$ from $P \cup Q_C \cup A$ exists. Conversely, if an SLD resolution exists, then we can assume that each ground instance of a rule is used only once, so an accepting run of our algorithm exists.

Since we are interested in data complexity, the number of predicates p and their arity r is bounded. Hence, if A contains c constants, we can describe each ground atom in $p \cdot r \cdot \lceil \log(c) \rceil$ bits. The number of atoms in G depends on the number of rules in $P \cup Q_C$, so storing G requires $k_1 \lceil \log(c) \rceil$ bits for k_1 a constant that does not depend on c. Finally, the number of ground clauses depends polynomially on c, so we can store the counter using $k_2 \lceil \log(c) \rceil$ bits for k_2 a constant that does not depend on c. Clearly, the algorithm requires $k \lceil \log(c) \rceil$ bits of space in total for k a constant that does not depend on c. The algorithm is nondeterministic, so it can be implemented in NLogSpace. □

We now apply Lemma 9 to show that answering conjunctive queries over DL-Lite$^+$ knowledge bases is NLogSpace-complete.

Theorem 2. *For a conjunctive query* $Q = \langle Q_P, \{Q_C\}\rangle$ *and a DL-Lite*$^+$ *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$*, deciding whether* $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ *is* NLogSpace*-complete w.r.t. data complexity.*

Proof. In [6], it was shown that checking entailment of a ground concept assertion is NLogSpace-hard if we allow for assertions of the form $\exists P.A \sqsubseteq B$. Membership follows immediately from Theorem 1, Lemmata 7 and 9, and the observation that the size of $\mathsf{rew}(Q, \mathcal{T})$ does not depend on the size of $\mathcal{A}$. □

By Lemma 8, if $\mathcal{T}$ is a DL-Lite TBox, then $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries, so we can compute answers to $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ in LogSpace with respect to data complexity [13], just as is the case in [7].

6 Conclusion

Motivated by the use of DL ontologies in Information Integration systems, we have presented a resolution-based algorithm for rewriting conjunctive queries over DL-Lite$^+$ TBoxes. We have used this algorithm to show that query answering in DL-Lite$^+$ can be implemented in NLogSpace w.r.t. data complexity. Together with the hardness result from [6], it follows that query answering in DL-Lite$^+$ is NLogSpace-complete with respect to data complexity,which closes what was, to the best of our knowledge, an open problem. Moreover, we have shown that our algorithm exhibits good "pay-as-you-go" behavior: on the subset of DL-Lite$^+$ for which query answering is in LogSpace, our algorithm is also worst-case optimal.

As part of our future work, we plan to extend the technique to deal with more expressive DLs, and in particular with an extended version of $\mathcal{EL}$; a sketch describing how this could be done was given at the end of Section 3. Such an algorithm would be optimal for the full spectrum of languages from DL-Lite to extended $\mathcal{EL}$—that is, languages for which the data complexity of query answering ranges from LogSpace to PTime-complete. Finally, we plan to implement our query answering technique in a prototype Information Integration system—we have established a promising relationship with researchers at the University of Newcastle who are using Information Integration in their ComparaGRID project,[1] and we plan to use ComparaGRID as an evaluation framework for our prototype system.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the EL Envelope. In: Proc. IJCAI 2005, Edinburgh, Scotland, pp. 364–369 (2005)
2. Baader, F., Nutt, W.: Basic Description Logics, ch. 2, pp. 47–100. Cambridge University Press, Cambridge (2003)

[1] http://www.comparagrid.org

3. Baader, F., Snyder, W.: Unification theory. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 8, vol. I, pp. 445–532. Elsevier Science, Amsterdam (2001)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch.2, vol. 1, pp. 19–100. North Holland, Amsterdam (2001)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: MASTRO-I: Efficient integration of relational data through DL ontologies. In: DL 2007 (2007)
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in Description Logics. In: Proc. DL 2005 (2005)
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning 9, 385–429 (2007)
8. Grädel, E.: Capturing complexity classes by fragments of second-order logic. Theor. Comput. Sci. 101, 35–57 (1992)
9. Green, C.: Theorem proving by resolution as a basis for question-answering systems. In: Meltzer, B., Michie, D. (eds.) 4th Annual Machine Intelligence Workshop, pp. 183–208. Edinburgh University Press (1969)
10. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal 10(4), 270–294 (2001)
11. Heflin, J., Hendler, J.: A portrait of the semantic web in action. IEEE Intelligent Systems 16(2), 54–59 (2001)
12. Joyner, W.H.: Resolution strategies as decision procedures. J. ACM 23(3), 398–417 (1976)
13. Kanellakis, P.C., Kuper, G.M., Revesz, P.Z.: Constraint query languages. In: Proc. PODS 1990, pp. 299–313 (1990)
14. Lee, J., Siau, K., Hong, S.: Enterprise integration with ERP and EAI. Commun. ACM 46(2), 54–60 (2003)
15. Lenzerini, M.: Data Integration: a theoretical perspective. In: Proc. PODS 2002, pp. 233–246. ACM Press, New York (2002)
16. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany (January 2006)
17. De Nivelle, H., Schmidt, R.A., Hustadt, U.: Resolution-Based Methods for Modal Logics. Logic Journal of the IGPL 8(3), 265–292 (2000)
18. Rosati, R.: On conjunctive query answering in EL. In: In Proceedings of CEUR Electronic Workshop, DL2007 (2007)

Author Index