Constraint-Based Approach to Semistructured Data*

Mohand-Saïd Hacid[1] Farouk Toumani[2] Ahmed K. Elmagarmid[1]

[1] Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 - USA
{*mshacid,ake*}*@cs.purdue.edu*

[2] Laboratoire LIMOS
ISIMA - Campus des Cezeaux - B.P. 125
63173 Aubière Cedex - France
ftoumani@sp.isima.fr

Abstract

In this paper we consider how constraint-based technology can be used to query semistructured data. As many concerns in semistructured data (e.g., representing and navigating) are also found in computational linguistics, this last area could provide an interesting angle to attack some of the problems regarding semistructured data. When possible, we should not duplicate works and good ideas should be reused. We present a formalism based on feature logics[1] for querying semistructured data. The formalism is a hybrid one in the sense that it combines clauses with path constraints. The resulting language has a clear declarative and operational semantics.

1 Introduction

In the last decade, new applications (e.g., CAD, CASE) have been a powerful driving force in the development of new database technology. New paradigms have arisen (e.g., object-oriented databases [10]) that provide greater flexibility than the traditional relational model. However, in some application areas, such as Web databases [4, 48], biological databases [66], digital libraries [21], etc; there is still a need for greater flexibility, both in data representation and manipulation. These applications are characterized by the lack of any fixed and rigid structure (i.e., schema).

Semistructured data models are intended to capture data that are not intentionally structured, that are structured heterogeneously, or that evolve so quickly that the changes cannot be reflected in the structure. A typical example is the World-Wide Web with its HTML pages, text files, bibliographies, biological databases, etc. A semistructured

*This project is supported by National Science Foundation under grants 9972883-EIA and 9974255-IIS and 9983249-EIA; grants from HP, IBM and Intel; and by the European Community, Basic Research Action IST-1999-10589 (Project MKBEEM).

[1] Feature logic (see, e.g., [11, 63, 32, 35]) has its origin in the three areas of knowledge representation with *concept descriptions*, *frames*, or Ψ*-terms* [6, 15, 51, 52], natural language processing, in particular approaches based on *unification grammars* [34, 36, 54, 62, 58, 60], and constraint (logic) programming [7, 8, 50, 64].

database essentially consists of objects, which are linked to each other by attributes.

The core problem in semistructured data is that the structure of the data is not fully known. This leads to the fact that querying the data is often content-based as opposed to the structure-based querying, e.g., in relational systems. Furthermore, this has led to the fact that users often browse through data instead, because no structural knowledge (or schema information) is available.

Semistructured Data represent a particularly interesting domain for query languages. Computations over semistructured data can easily become infinite, even when the underlying alphabet is finite. This is because the use of path expressions (i.e., compositions of labels) is allowed, so that the number of possible paths over any finite alphabet is infinite. Query languages for semistructured data have been recently investigated mainly in the context of algebraic programming [3, 17].

In this paper, we explore a different approach to the problem, an approach based on logic programming, instead of algebraic programming. In particular, we develop an extension of *Datalog* for manipulating semistructured data. The resulting language has both a clear declarative semantics and an operational semantics. The semantics are based on fixpoint theory, as in classical logic programming [42]. The language of terms uses five countable, disjoint sets: a set of atomic values ($\mathcal{D}_1$), a set of objects ($\mathcal{D}_2$), a set of labels ($\mathcal{D}_3$), a set of object variables ($\mathcal{V}$), and a set of path variables ($\tilde{V}$). A path variable is a variable ranging over paths. The universe of paths over $\mathcal{D}_3$ is infinite. Thus, to keep the semantics of programs finite, we do not evaluate rules over the entire universe, $\mathcal{D}_3^*$, but on a specific *active domain*. We define the *active domain* of a database to be the set of constants (objects and labels) occurring in the database. We then define the *extended active domain*[2] to include both the constants in the *active domain* and all path expressions resulting from the composition of labels in the *active domain*. The semantics of our language is defined with respect to the *extended active domain*. In particular, substitutions range over this domain when rules are evaluated.

The *extended active domain* is not fixed during query evaluation. Instead, whenever a new path expression is created, the new path and all paths resulting from its concatenation with already existing paths are added to the *extended active domain*.

We present a class of path constraints of interest in connection with both structured and semistructured databases. Our constraint language is inspired by Feature Logics [12]. Feature descriptions are used as the main data structure of so-called unification grammars, which are a popular family of declarative formalisms for processing natural language [62]. Feature descriptions have also been proposed as a constraint system for logic programming (see, for example, [9, 65]). They provide for a partial description of abstract objects by means of functional attributes called features. On top of this constraint language we allow the definition of relations (by means of definite clauses) in the style of [31], leading to a

[2]This notion is borrowed from [45].

declarative, rule-based, constraint query language for semistructured data. The language we propose is based on the general scheme for handling clauses whose variables are constrained by an underlying constraint theory [19]. Constraints can be seen as quantifier restrictions as they filter out the values that can be assigned to the variables of a clause in any of the models of the constraint theory. Containment[3] of queries is the problem of checking whether the result of one query is contained in what another query produces [2, 67]. Containment is mainly concerned with query optimization. In this paper we propose an approach for testing containment of queries expressed in our rule-based, constraint language.

To our knowledge (see related work in Section 8), no previous work considers path constraints we can express in our query language.

Paper outline: Section 2 introduces semistructured data. Section 3 gives a brief summary of Feature Logics. Section 4 defines preliminary notions such as the Resolution Principle for clauses with constraints. In Section 5 we introduce the data model and give an introductory example. In Section 6, we develop our language and give its syntax and semantics. In Section 7 we develop a calculus for deciding query containment. Section 8 discusses related work. We conclude in Section 9.

2 Semistructured Data

In traditional databases such as the relational model [24] there is a clear separation between the schema and the data itself. Recently, it has been recognized that there are applications where the data is self-describing in the sense that it does not come with a separate schema, and the structure of the data, when it exists has to be inferred from the data itself. Such data is called *semistructured*. A concrete example is the ACeDB genome database [66], while a somewhat less concrete but certainly well-known example is the World-Wide Web. The Web imposes no constraints on the internal structure of HTML pages, although structural primitives such as enumerations may be used. Another frequent scenario for semistructured data is when data is integrated in a simple fashion from several heterogeneous sources and there are discrepancies among the various data representations: some information may be missing in some sources, an attribute may be single-valued in one source and multi-valued in another, or the same entity may be represented by different types in different sources. Semistructured data displays the following features:

> *The structure is irregular:* In many applications, the large collections that are maintained often consist of heterogeneous elements. Some elements may be incomplete. On the other hand, other elements may record extra information, e.g., annotations. Different types may be used for the same kind of information, e.g., prices may be in dollars in portions of the database and in francs in others. The same piece of infor-

[3] Also called implication.

mation, e.g., an address, may be structured in some places as a string and in others as a tuple. Modeling and querying such irregular structures are essential issues.

The structure is implicit: In many applications, although a precise structuring exists, it is given implicitly. For instance, electronic documents consist of a text and a grammar (e.g., a DTD in SGML). The parsing of the document then allows one to isolate pieces of information and detect relationships between them. However, the interpretation of these relationships (e.g., SGML exceptions) may be beyond the capabilities of standard database models and are left to the particular applications and specific tools.

Data models, query languages, and systems for semistructured data are areas of active research. Of particular interest and relevance, eXensible Markup Language (XML) is an emerging standard for web data, and bears a close correspondence to semistructured data models introduced in research. Semistructured data is naturally modeled in terms of graphs which contain labels that give semantics to the underlying structure [1, 16].

Management of semistructured data requires typical database features such as a language for forming adhoc queries and updates, concurrency control, secondary storage management, etc. However, because semistructured data cannot conform to a standard database framework, trying to use a conventional DBMS to manage semistructured data becomes a difficult or impossible task.

When querying semistructured data, one cannot expect the user to be fully aware of the complete structure, especially if the structure evolves dynamically. Thus, it is important not to require full knowledge of the structure to express meaningful queries. At the same time, we do not want to be able to exploit regular structure during query processing when it happens to exist and the user happens to know it.

An example of a complete database management system for semistructured data is LORE [44], a repository for OEM [69] data featuring the LOREL [3] query language. Another system devoted to semistructured data is the Strudel Web site management system, which features the StruQL query language [27] and a data model similar to OEM. UnQL [17] is a query language that allows queries on both the content and structure of a semistructured database and also uses a data model similar to Strudel and OEM.

3 Feature Logics

Feature logics are the logical basis for so-called unification grammars studied in computational linguistics. The common assumption of these formalisms is that linguistic objects can be described by means of their features, which are functional attributes.

Constraint-Based grammar Formalisms

The constraint-based grammar formalisms have evolved relatively independently, and accordingly been shaped by rather different design objectives and theoretical positions. On the one hand, unification-based formalisms such as PATR [60, 61] and FUG [36, 37] were designed primarily as general, computational tools to facilitate the task of implementing natural language processing systems. As such they impose few (non-trivial) constraints on the kinds of linguistic descriptions which can be written down. Formalisms such as LFG [34], on the other hand, were designed to serve as formal realizations of particular theoretical frameworks, and accordingly embody restrictive statements of "universal grammar". Despite their different origins, and their distinctive and often innovatory features however, the constraint-based grammar formalisms share a number of important characteristics such as:

Declarativism. The constraint-based grammar formalisms characterize the relationship between natural language expressions and the information they carry (both syntactic and semantic) without reference to particular models of linguistic processing.

Surface Orientation/Context-Free Skeleton. Constraint-based grammar formalisms provide a direct characterization of the surface form of natural language expressions. Typically, they make use of a context-free 'skeleton' grammar (either phrase-structure rules or a categorical component) in order to describe the hierarchical structure and left-to-right order of strings.

Feature Structures. An important characteristic of constraint-based grammar formalisms is the use of hierarchical systems of features and values known as feature structures to represent linguistic objects. Feature structures are related to the record structures of computer science and the frames of Artificial Intelligence.

Constraint Languages. In order to characterize the well-formedness of linguistic objects, constraint-based grammar formalisms employ specialized languages for expressing propositions about features and their values. Constraint languages are used to capture syntactic phenomena such as agreement, subcategorization and unbounded dependencies, and account for much of the notational elegance of the formalisms.

Feature Constraints

Feature constraints provide records as logical data structure for constraint (logic) programming. Their origins are the feature descriptions from computational linguistics (see [63] for references) and Aït-Kaci's ψ-terms [6] which have been employed in the logic programming language LOGIN [7]. Smolka [63] gives a unified logical view of most earlier feature formalisms and presents an expressive feature logic.

The predicate logic view to feature constraints, which has been pioneered by Smolka [63], laid the ground for the development of the constraint system FT [9] and CFT [65]. The

latter constraint system subsumes Colmerauer's classical rational tree constraint system [25], but provides for finer grained and more expressive constraints. CFT's standard model consists of so-called feature trees, that is possibly infinite trees where the nodes are labeled with label symbols and the edges are labeled with feature symbols. The labels of the edges departing from a node, called the features of that node, are pairwise distinct.The atomic constraints of CFT are equations, label constraints Ax ("x has label A"), feature constraints $x[f]y$ ("y is a child of x via feature f") and arity constraints $x\{f_1,\ldots,f_n\}$ ("x has exactly the features $f_1,\ldots,f_n$"). A rational tree constraint $x\doteq K(y_1,\ldots,y_n)$ can be expressed in CFT as

$$Kx \wedge x\{1,\ldots,n\} \wedge x[1]y_1 \wedge \ldots x[n]y_n$$

Note that in CFT we can express the fact that x has the child y at feature f by $x[f]y$, which is inconvenient to express in the rational tree constraint system if the signature is finite and impossible if the signature is infinite.

4 Preliminaries

We shall introduce those notions that are necessary for our application (see, e.g., [19, 31, 40, 43] for further details).

A *constraint* is some piece of syntax constraining the values of the variables occurring in it. It is a restriction on a space of possibilities. Mathematical constraints are precisely specifiable relations among several unknowns (or variables), each taking a value in a given domain. Constraints restrict the possible values that variables can take, representing some (partial) information about the variables of interest. Constraints enjoy several interesting properties. First, as said before, constraints may specify partial information –a constraint need not uniquely specify the value of its variables. Second, they are additive: given a constraint c_1 say, $X + Y \geq Z$, another constraint c_2 can be added, say, $X + Y \leq Z$. The order of imposition of constraints does not matter; all that matters at the end is that the conjunction of constraints is in effect. Third, constraints are rarely independent; for instance, once c_1 and c_2 are imposed it is the case that the constraint $X + Y = Z$ is entailed. Fourth, they are nondirectional: typically a constraint on (say) three variables X, Y, Z can be used to infer a constraint on X given constraints on Y and Z, or a constraint on Y given constraints on X and Z and so on. Fifth, they are declarative: typically they specify what relationship must hold without specifying a computational procedure to enforce that relationship. Any computational system dealing with constraints must fundamentally take these properties into account.

Now, let us formally define constraint languages.

A constraint language is a tuple $(VAR, CON, \mathcal{V}, INT)$ such that:

1. VAR is an infinite set whose elements are called variables
2. CON is a set whose elements are called constraints

3. $\mathcal{V}$ is a computable function that assigns to every constraint ϕ a finite set $\mathcal{V}\phi$ of variables, called the variables constrained by ϕ

4. INT is a nonempty set of **interpretations**, where every interpretation $\mathcal{I} \in INT$ consists of a nonempty set $D^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a solution mapping $[\cdot]^{\mathcal{I}}$ such that:

   4.1 an $\mathcal{I}$-assignment is a mapping $VAR \longrightarrow D^{\mathcal{I}}$, and $ASS^{\mathcal{I}}$ is the set of all $\mathcal{I}$-assignments

   4.2 $[\cdot]^{\mathcal{I}}$ is a function mapping every constraint ϕ to a set $[\phi]^{\mathcal{I}}$ of $\mathcal{I}$-assignments, where the $\mathcal{I}$-assignments in $[\phi]^{\mathcal{I}}$ are called the solutions of ϕ in $\mathcal{I}$

   4.3 a constraint ϕ contains only the variables in $\mathcal{V}\phi$, that is, if $\alpha \in [\phi]^{\mathcal{I}}$ and β is an $\mathcal{I}$-assignment that agrees with α on $\mathcal{V}\phi$, then $\beta \in [\phi]^{\mathcal{I}}$

Predicate logic is a prominent example of a constraint language. The well-formed formulas are the constraints, $\mathcal{V}\phi$ are the free variables of a formula ϕ, and for every Tarski interpretation $\mathcal{I}$ the solutions $[\phi]^{\mathcal{I}}$ are the $\mathcal{I}$-assignments satisfying ϕ.Viewing predicate logic as a constraint language abstracts away from the syntactic details of formulas.

A constraint ϕ is satisfiable if there exists at least one interpretation in which ϕ has a solution. A constraint ϕ is valid in an interpretation $\mathcal{I}$ if $[\phi]^{\mathcal{I}} = ASS^{\mathcal{I}}$, that is, every $\mathcal{I}$-assignment is a solution of ϕ in $\mathcal{I}$. Conversely, we say that an interpretation $\mathcal{I}$ satisfies a constraint ϕ if ϕ is valid in $\mathcal{I}$. An interpretation is a model of a set Φ of constraints if it satisfies every constraint in Φ.

A renaming is a bijection $VAR \to VAR$ that is the identity except for finitely many exceptions. If ρ is a renaming, we call a constraint ϕ' a ρ-variant of a constraint ϕ if

$$\mathcal{V}\phi' = \rho(\mathcal{V}\phi) \text{ and } [\phi]^{\mathcal{I}} = [\phi']^{\mathcal{I}} \qquad \rho := \{\alpha\rho \mid \alpha \in [\phi']^{\mathcal{I}}\}$$

for every interpretation $\mathcal{I}$. A constraint ϕ' is called a variant of a constraint ϕ if there exists a renaming ρ such that ϕ' is a ρ-variant of ϕ.

Proposition 1 *A constraint is satisfiable if and only if each of its variants is satisfiable. Furthermore, a constraint is valid in an interpretation $\mathcal{I}$ if and only if each of its variants is valid in $\mathcal{I}$.*

A constraint language is closed under renaming if every constraint ϕ has a ρ-variant for every renaming. A constraint language is closed under intersection if for every two constraints ϕ and ϕ' there exists a constraint ψ such that $[\phi]^{\mathcal{I}} \cap [\phi']^{\mathcal{I}} = [\psi]^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. A constraint language is decidable if the satisfiability of its constraints is decidable.

Let Φ be a set of constraints and $\mathcal{I}$ be an interpretation. The solutions of Φ in $\mathcal{I}$ are defined as

$$[\Phi]^{\mathcal{I}} := \bigcup_{\phi \in \Phi} [\phi]^{\mathcal{I}}$$

where $[\Phi]^{\mathcal{I}} := \emptyset$ is Φ is empty.

Now, we introduce a general schema for handling clauses whose variables are constrained with an underlying constraint theory. In general, constraints can be seen as quantifier restrictions as they filter out the values that can be assigned to the variables of a clause (or an arbitrary formulae with restricted universal or existential quantifier) in any of the models of the constraint theory. We give a resolution principle for clauses with constraints, where unification is replaced by testing constraints for satisfiability over the constraint theory.

The Resolution Principle is based on the following inference rule:

$$\text{From } (A \vee B) \text{ and } (\neg A \vee C) \text{ infer } (B \vee C)$$

The rule is obviously correct: If $(A \vee B)$ and $(\neg A \vee C)$ are true, then the resolvent $(B \vee C)$ is also true. Since either A is false, then B must be true, or A is true, then C must be true, and hence in any case $(B \vee C)$ must be true.

For predicate logic the two complementary A's are atoms starting with the same predicate symbol, but with potentially different argument terms, say s_i and t_i $(i \in [1, n])$. Then one has to *unify* corresponding arguments of the literals A and $\neg A$ –that is to find substitutions for the variables that make the corresponding argument terms identical– before the conclusion can be drawn, and the resolvent has to be instantiated with the unifying substitution:

$$\frac{P(s_1, \ldots, s_n) \vee B \ \textit{and} \ \neg P(t_1, \ldots, t_n) \vee C}{\sigma(B \vee C)} \qquad \text{if } \sigma s_i = \sigma t_i (1 \leq i \leq n)$$

Robinson [57] showed that this rule (combined with factorization) provides a complete calculus. An analysis of the soundness and completeness proof, however, shows that it is not necessary to unify the terms. It would be enough to test whether they are unifiable, provided we add a constraint Γ consisting of term equations $s_i = t_i$ (saying "if the terms can be made equal") to the inferred resolvent. Whenever such constrained resolvents are now involved in a further resolution step the new resolvent inherits their constraints together with the new argument term equations. Now, these collected constraints have to be tested for unifiability. Thus a resolution step in this modification takes two clauses with such equational constraints and produces a resolvent with a new unifiable equational constraint consisting of the constraints inherited from its parents together with the argument equations for the involved complementary literals.

$$\frac{P(s_1, \ldots, s_n) \vee B \mid\mid \Gamma \ \textit{and} \ \neg P(t_1, \ldots, t_n) \vee C \mid\mid \Delta}{B \vee C \mid\mid \Gamma \wedge \Delta \wedge s_i = t_i} \qquad \text{if } \Gamma \wedge \Delta \wedge s_i = t_i \text{ is } \textit{unifiable}$$

A more general view is that every clause might have some arbitrary, not necessarily equational constraint and a resolvent of two clauses gets a new constraint that is unsatisfiable

whenever one of the constraints of the parents (or the equational constraint of the arguments of the complementary literals) is unifiable.

$$\frac{P(s_1,\ldots,s_n) \vee B \;||\; R \text{ and } \neg P(t_1,\ldots,t_n) \vee C \;||\; S}{B \vee C \;||\; R \wedge S \wedge s_i = t_i} \qquad \text{if } R \wedge S \wedge s_i = t_i \text{ is } \textit{satisfiable}$$

This constrained resolution principle is again sound, but as in the classical case completeness of a constrained resolution calculus is not straight forward.

5 Data Model and Queries

Recent research works propose to model semistructured data using *"lightweight"* data models based on labeled directed graphs [1, 17]. Informally, the vertices in such graphs represent objects and the labels on the edges convey semantic information about the relationship between objects. The vertices without outgoing edges (sink nodes) in the graph represent atomic objects and have values associated with them. The other vertices represent complex objects. An example[4] of a semistructured database is given figure 1. Although a real-world video database would of course be much, much larger, this example concisely captures the sort of structure (or lack thereof) needed to illustrate the features of our language. As illustrated by figure 1, the structure of the content describing a video differs from a category to another, and even within the same category.

Path expressions describe path along the graph, and can be viewed as compositions of labels. For example, the expression

Residence.City

describes a path that starts in an object, continues to the residence of that object, and ends in the city of that residence.

In this paper, we assume that the usual base types String, Integer, Real, etc., are available. In addition, we shall use a new type *Feature* for labels that would correspond to attribute (i.e., label) names. We write numbers and labels literally and use quotation marks for strings, e.g., "car". In what follows we make the simplifying assumption that labels can be symbols, strings, integers, etc; in fact, the type of labels is just the discriminated union of these base types.

In the following, we restrict ourselves to semistructured data whose graph is **acyclic**.

Like [53] we represent the graph using two base relations:

- *link(FromObj, ToObj, Label)*: This relation contains all the edge information. For instance, *link*(*archive-entry*, o_1, *video*) refers to an edge labeled *video* from the object *archive-entry* to the object o_1.

[4] This example is inspired by the one given in [30].

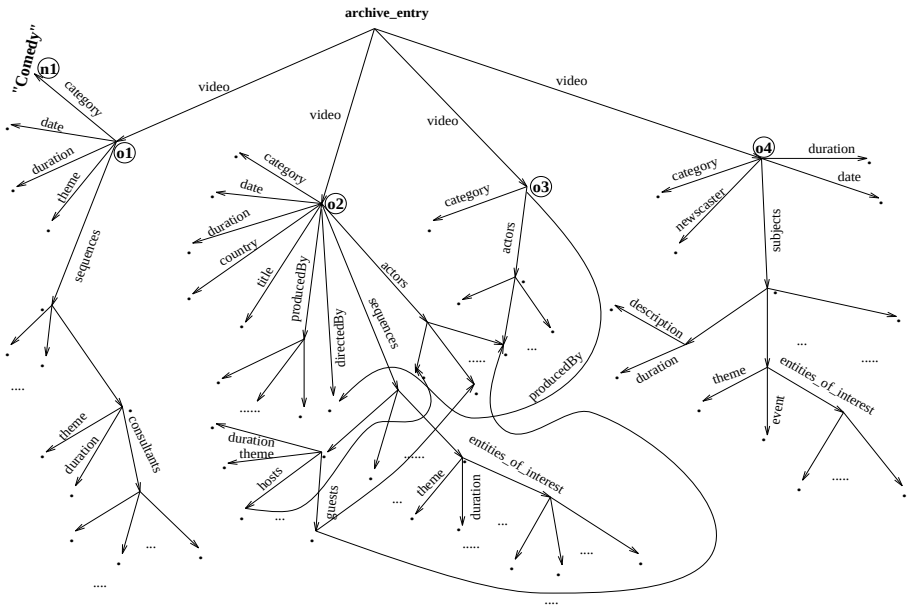


Figure 1: A video database content.

- *atomic(Obj, Value)*: It contains all the value information. For instance, the fact *atomic*(n_1,"*Comedy*") states that object n_1 is atomic and has value *Comedy*.

We assume that each atomic object has exactly one value, and each atomic object has no outgoing edges. We consider that the data comes in as an instance over *link* and *atomic* satisfying these two conditions. We use the term database in the following for such a data set.

Let p be a path expression of the form $a_1.a_2 \ldots a_n$, where each a_i is a label. Then, *link*(o_1, o_2, p) holds if there are objects $o_1^{'}, \ldots, o_{n-1}^{'}$ such that *link*(o_1, $o_1^{'}$, a_1), ..., *link*($o_{n-1}^{'}$, o_2, a_n) hold.

The following are examples of queries (over the database of figure 1). In these queries, X and Y are first-order variables, and α, β are path variables, that is variables ranging over paths built from an alphabet of labels.

$$\mathsf{Answer(X)} : -\mathsf{link(archive\text{-}entry, X}, \alpha) \| \alpha \dot{\in} \mathsf{video.actor}$$

This query returns the set of objects reachable from the root object archive-entry by the path video.actor. The symbol $\dot{\in}$ is used to express path restriction. In this query, Answer(X) is called the head of the query, link(archive-entry, X, α) is called the body of the query, and $\alpha \dot{\in}$video.actor is called the constraint part of the query.

$$\mathsf{Answer(X, Y)} : -\mathsf{link(X, Y}, \alpha), \mathsf{link(X, Y}, \beta) \| \alpha \dot{\amalg} \beta$$

This query returns a set of pairs (X, Y) of objects, such that X is an ancestor of Y via two different paths. The symbol $\dot{\amalg}$ stands for divergence of paths.

$$\mathsf{Answer(X,Y)} : -\mathsf{link(X,Y,}\alpha\mathsf{)} \| \beta \dot{\prec} \alpha, \beta \dot{\in} \mathsf{video.sequences}$$

This query retrieves a set of pairs (X, Y) such that Y is reachable from X by a path starting with the sub-path video.sequences and ending with any sub-path. The symbol $\dot{\prec}$ is used to express a prefix relation between two paths.

6 The Language

We view our language as consisting of a constraint language on top of which relations can be defined by definite clauses.

6.1 Basic Definitions

Before providing the syntax and semantics of our query language, we need some additional definitions.

Definition 1 *(**Path**) A path is a finite string of labels. We identify a label f with the string (f) consisting of a single label. We say that a path u is a* **prefix** *of a path v (written $u \prec v$) if there is a non-empty path w such that $v = u.w$. Note that $\prec$ is neither symmetric nor reflexive. We say that two paths u, v* **diverge** *(written $u \amalg v$) if there are labels f, g with $f \neq g$, and possibly empty paths w, w_1, w_2, such that $u = w.f.w_1 \wedge v = w.g.w_2$. It is clear that $\amalg$ is a symmetric relation.*

Proposition 2 *Given two paths u and v, then exactly one of the relations $u = v$, $u \prec v$, $u \succ v$ or $u \amalg v$ holds.*

Definition 2 *(**Path Term**) A path term, denoted p, q, ..., is either a path variable α or a concatenation of path variables $\alpha.\beta$.*

6.2 Syntax

The language of terms uses three countable, pair-wise disjoint sets:

1. A set $\mathcal{D}$ of constant symbols. This set is the union of three pair-wise disjoint sets:
   - $\mathcal{D}_1$: a set of atomic values
   - $\mathcal{D}_2$: a set entities, also called objects entities
   - $\mathcal{D}_3$: a set of labels
2. A set $\mathcal{V}$ of variables called object variables and value variables, and denoted $X, Y, \ldots$
3. A set $\tilde{V}$ of variables called path variables, and denoted $\alpha, \beta, \ldots$

We call $\mathcal{D} \cup \mathcal{V}$ first order terms.

6.3 Constraint Language ($\mathcal{C}$)

We start with a definition of our constraint language.

Definition 3 (Path Constraint) *The set of atomic constraints is given by:*

c	$\longrightarrow$	$X \doteq Y$	**agreement**
		$\alpha \dot{\prec} \beta$	**prefix**
		$p \dot{\in} L$	**path restriction**
		$\alpha \doteq \beta$	**path equality**
		$p \dot{\amalg} q$	**divergence**

We exclude empty paths in subterm agreements, since $X \epsilon Y$ is equivalent to $X \doteq Y$. L is a regular expression denoting a regular language $\mathcal{L}(L) \subseteq \mathcal{F}^+$, where $\mathcal{F}$ is a set of labels. Note that, to guarantee the decidability of conjunctions of constraints, complex terms are allowed only in divergence and restriction constraints.

An **interpretation** $\mathcal{I}$ is a standard first order structure, where every label $f \in \mathcal{F}$ is interpreted as a binary, functional relation $f^{\mathcal{I}}$. A **valuation** is a pair $(\nu_{\mathcal{V}}, \nu_{\tilde{V}})$, where $\nu_{\mathcal{V}}$ is a standard first order valuation of the variables in $\mathcal{V}$ and $\nu_{\tilde{V}}$ is a function $\nu_{\tilde{V}} : \tilde{V} \to \mathcal{F}^+$. We define $\nu_{\tilde{V}}(\alpha.\beta)$ to be $\nu_{\tilde{V}}(\alpha)\nu_{\tilde{V}}(\beta)$.

The **validity** of an atomic constraint in an interpretation $\mathcal{I}$ under a valuation $(\nu_{\mathcal{V}}, \nu_{\tilde{V}})$ is defined as follows:

$$
\begin{array}{lcl}
(\nu_{\mathcal{V}}, \nu_{\tilde{V}}) \models_{\mathcal{I}} X \doteq Y & \Longleftrightarrow & \nu_{\mathcal{V}}(X) = \nu_{\mathcal{V}}(Y) \\
(\nu_{\mathcal{V}}, \nu_{\tilde{V}}) \models_{\mathcal{I}} p \dot{\in} L & \Longleftrightarrow & \nu_{\tilde{V}}(p) \in L \\
(\nu_{\mathcal{V}}, \nu_{\tilde{V}}) \models_{\mathcal{I}} \alpha \dot{\prec} \beta & \Longleftrightarrow & \nu_{\tilde{V}}(\alpha) \prec \nu_{\tilde{V}}(\beta) \\
(\nu_{\mathcal{V}}, \nu_{\tilde{V}}) \models_{\mathcal{I}} \alpha \doteq \beta & \Longleftrightarrow & \nu_{\tilde{V}}(\alpha) = \nu_{\tilde{V}}(\beta) \\
(\nu_{\mathcal{V}}, \nu_{\tilde{V}}) \models_{\mathcal{I}} p \dot{\amalg} q & \Longleftrightarrow & \nu_{\tilde{V}}(p) \amalg \nu_{\tilde{V}}(q)
\end{array}
$$

Note that there is no interaction between $\nu_{\mathcal{V}}$ and $\nu_{\tilde{V}}$.

A constraint ϕ is **satisfiable** if there exists at least one interpretation in which ϕ has a solution. **Satisfiability** of conjunctions of atomic constraints is **decidable** [12].

6.4 Relational Extension

We now present a construction that, given the constraint language $\mathcal{C}$ and a set of relation symbols, extends $\mathcal{C}$ to a constraint language $\mathcal{R}(\mathcal{C})$.

Definition 4 (Predicate Symbol) *we define the following predicate symbols:*

- *The predicate symbol link with arity 3*
- *The binary predicate symbol atomic*

- *The user specified intentional predicates (ordinary[5] predicates)*

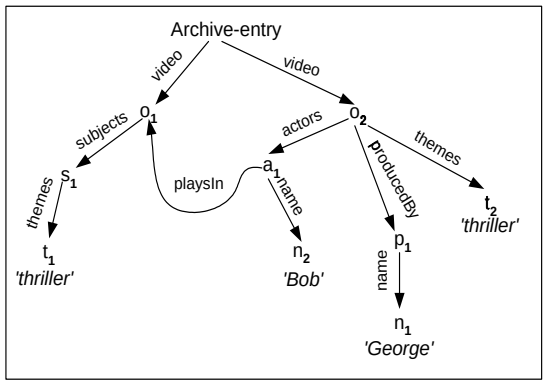
We model semistructured data by a program P which contains, besides the set of facts built from link and atomic, the following rule:

$$link(X, Y, \alpha.\beta) : -link(X, Z, \alpha), link(Z, Y, \beta)$$

This rule says that if an object Z can be reached from an object X by a path α and Y can be reached from Z by the path β, then there is a path $\alpha.\beta$ from X to Y.
Other ordinary predicates can be specified by rules of the form:

$$H(\bar{X}) : -L_1(\bar{Y}_1), \ldots, L_n(\bar{Y}_n) \| c_1, \ldots, c_m$$

for some $n \geq 0$ and $m \geq 0$, where $\bar{X}, \bar{Y}_1, \ldots, \bar{Y}_n$ are tuples of variables or constants. We require that the rules are safe, i.e., a variable that appears in $\bar{X}$ must also appear in $\bar{Y}_1 \cup \ldots \cup \bar{Y}_n$. The predicates $L_1, \ldots, L_n$ may be either *link* or *atomic*, or ordinary predicates. $c_1, \ldots, cm$ are path constraints ($\mathcal{C}$-constraints). In the following, we use the term (positive) atom to make reference to predicates $L_1, \ldots, L_n$.
Figure 2 shows a program associated with a fragment of the video database of Figure 1. This program P does not contain arbitrary predicates.



The program P

atomic(n_1, 'George'),
atomic(n_2, 'Bob'),
atomic(t_1, 'thriller'),
atomic(t_2, 'thriller')
link(**archive-entry**, o_1, video),
link(**archive-entry**, o_2, video),
link(o_2, a_1, actors),
link(o_1, s_1, subjects),
link(o_2, t_2, themes),
link(s_1, t_1, themes),
link(a_1, o_1, playsIn),
link(o_2, p_1, producedBy),
link(p_1, n_1, name),
link(a_1, n_2, name),

link(X, Y, α.β) :- link(X, Z, α), link(Z, Y, β)

Figure 2: A fragment of a video database and its corresponding program P.

6.5 Semantics

Our language has a declarative model-theoretic and fixpoint semantics.

Model-theoretic semantics Recall that $\mathcal{V}$ denotes a set of variables called object and value variables, and $\tilde{V}$ denotes a set of variables called path variables. Let $\hat{V} = \mathcal{V} \cup \tilde{V}$.
Let var_1 be a countable function that assigns to each syntactical expression a subset of $\mathcal{V}$ corresponding to the set of object and value variables occurring in the expression, and var_2 be a countable function that assigns to each expression a subset of $\tilde{V}$ corresponding

[5]Ordinary predicates are of any arity.

to the set of path variables occurring in the expression.

Let $var = var_1 \cup var_2$. If $E_1, \ldots, E_n$ are syntactic expressions, then $var(E_1, \ldots, E_n)$ is an abbreviation for $var(E_1) \cup \ldots \cup var(E_n)$.

A ground atom A is an atom for which $var(A) = \emptyset$. A ground rule is a rule r for which $var(r) = \emptyset$.

Definition 5 (Extension) *Given a set $\mathcal{D}_3$ of labels, the extension of $\mathcal{D}_3$, written $\mathcal{D}_3^{ext}$, is the set of path expressions containing the following elements:*

- *each element in $\mathcal{D}_3$*
- *for each ordered pair p_1, p_2 of elements of $\mathcal{D}_3^{ext}$, the element $p_1.p_2$*

Definition 6 (Extended Active Domain) *The active domain of an interpretation $\mathcal{I}$, noted $\mathcal{D}_\mathcal{I}$ is the set of elements appearing in $\mathcal{I}$, that is, a subset of $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$. The extended active domain of $\mathcal{I}$, denoted $\mathcal{D}_\mathcal{I}^{ext}$, is the extension of $\mathcal{D}_\mathcal{I}$, that is, a subset of $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3^{ext}$.*

Definition 7 (Interpretation) *Given a program P, an interpretation $\mathcal{I}$ of P consists of:*

- *A domain $\mathcal{D}$*
- *A mapping from each constant symbol in P to an element of domain $\mathcal{D}$*
- *A mapping from each n-ary predicate symbol in P to a relation in $(\mathcal{D}^{ext})^n$*

Definition 8 (Valuation) *A valuation ν_1 is a total function from $\mathcal{V}$ to the set of elements $\mathcal{D}_1 \cup \mathcal{D}_2$. A valuation ν_2 is a total function from $\tilde{V}$ to the set of elements $\mathcal{D}_3^{ext}$. Let $\nu = \nu_1 \cup \nu_2$. ν is extended to be identity on $\mathcal{D}$ and then extended to map free tuples to tuples in a natural fashion.*

Definition 9 (Atom Satisfaction) *Let $\mathcal{I}$ be an interpretation. A ground atom L is satisfiable in $\mathcal{I}$ if L is present in $\mathcal{I}$.*

Definition 10 (Rule Satisfaction) *Let r be a rule of the form :*

$$r : A \leftarrow L_1, \ldots, L_n \| c_1, \ldots, c_m$$

where $L_1, \ldots, L_n$ are (positive) atoms, and $c_1, \ldots, c_m$ are path constraints. Let $\mathcal{I}$ be an interpretation, and ν be a valuation that maps all variables of r to elements of $\mathcal{D}_\mathcal{I}^{ext}$. The rule r is said to be true (or satisfied) in interpretation $\mathcal{I}$ for valuation ν if $\nu[A]$ is present in $\mathcal{I}$ whenever each $\nu[L_i], i \in [1, n]$ is satisfiable in $\mathcal{I}$, and each $\nu[c_j], j \in [1, m]$ is satisfiable.

6.5.1 Fixpoint Semantics

The fixpoint semantics is defined in terms of an immediate consequence operator, T_P, that maps interpretations to interpretations. An interpretation of a program is any subset of all ground atomic formulas built from predicate symbols in the language and elements in $\mathcal{D}^{ext}$.
Each application of the operator T_P may create new atoms. We show below that T_P is monotonic and continuous. Hence, it has a least fixpoint that can be computed in a bottom-up iterative fashion.

Recall that the language of terms has three countable disjoint sets: a set of atomic values ($\mathcal{D}_1$), a set of entities ($\mathcal{D}_2$), and a set of labels ($\mathcal{D}_3$). A path expression is an element of $\mathcal{D}_3^{ext}$. We define $\mathcal{D}^{ext} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3^{ext}$.

At the iteration 1, only paths of length[6] 1 (i.e., simple labels, elements of $\mathcal{D}_3$) are considered during rules triggering. At iteration k, we consider paths of length less or equal to k (cf. figure 3). It is clear that the paths occurring in the final result are of length less or equal to the size (i.e., cardinality) of the base relation *link*.

Lemma 1 *If $\mathcal{I}_1$ and $\mathcal{I}_2$ are two interpretations such that $\mathcal{I}_1 \subseteq \mathcal{I}_2$, then $\mathcal{D}_{\mathcal{I}_1}^{ext} \subseteq \mathcal{D}_{\mathcal{I}_2}^{ext}$.*

Definition 11 (Immediate Consequence Operator) *Let P be a program and $\mathcal{I}$ an interpretation. A ground atom A is an immediate consequence for $\mathcal{I}$ and P if either $A \in \mathcal{I}$, or there exists a rule $r : H \leftarrow L_1, \ldots, L_n \| c_1, \ldots, c_m$ in P, and there exists a valuation ν, based on $\mathcal{D}_{\mathcal{I}}^{ext}$, such that:*

- *$A = \nu(H)$, and*
- *$\forall i \in [1, n]$, $\nu(L_i)$ is satisfiable, and*
- *$\nu(c_1, \ldots, c_m)$ satisfiable.*

Definition 12 (T-Operator) *The operator T_P associated with program P maps interpretations to interpretations. If $\mathcal{I}$ is an interpretation, then $T_P(\mathcal{I})$ is the following interpretation:*

$$T_P(\mathcal{I}) = \mathcal{I} \cup \{A \mid A \text{ is an immediate consequence for } \mathcal{I} \text{ and } P\}$$

Figure 3 shows the new facts produced at each iteration of the operator T_P in the case of the program P of Figure 2.
Note that, after the iteration 5, no new facts can be produced.

Lemma 2 (Monotonicity) *The operator T_P is monotonic; i.e., if $\mathcal{I}_1$ and $\mathcal{I}_2$ are two interpretations such that $\mathcal{I}_1 \subseteq \mathcal{I}_2$, then $T_P(\mathcal{I}_1) \subseteq T_P(\mathcal{I}_2)$*

[6]The length of a path is the number of labels composing the path.

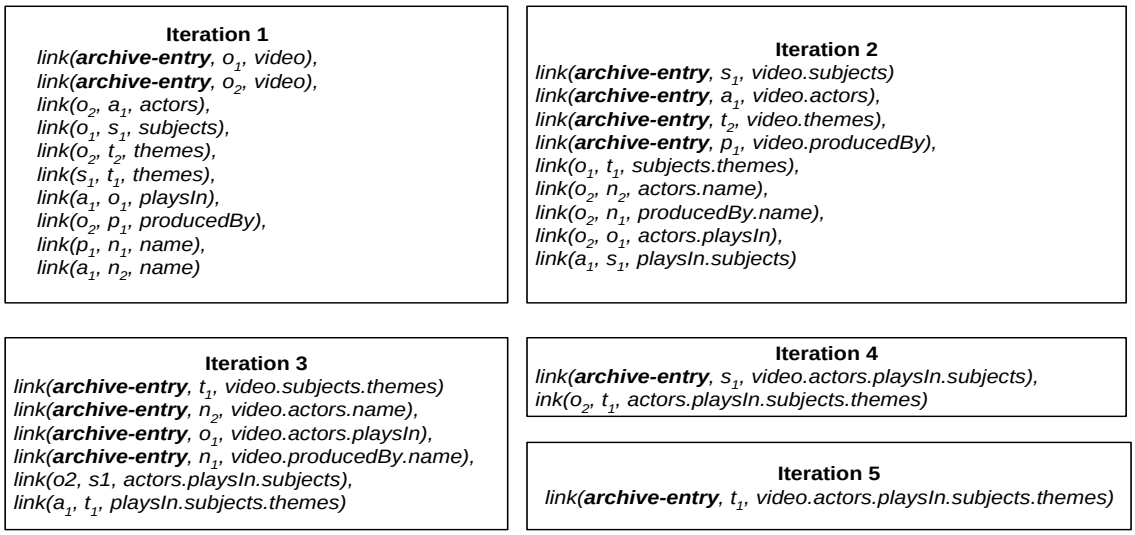


Figure 3: The new facts (immediate consequences) produced at each iteration.

Proof Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two interpretations such that $\mathcal{I}_1 \subseteq \mathcal{I}_2$. We must show that if an atom A i s an immediate consequence for $\mathcal{I}_1$ and P, then $A \in T_P(\mathcal{I}_2)$.
Since A is an immediate consequence for $\mathcal{I}_1$ and P, at least one of the following cases applies:

- $A \in \mathcal{I}_1$. Then $A \in \mathcal{I}_2$, and thus $A \in T_P(\mathcal{I}_2)$;
- there exists a rule $r : H \leftarrow L_1, \ldots, L_n \| c_1, \ldots, c_m$ in P and a valuation ν, based on $\mathcal{D}^{ext}_{\mathcal{I}_1}$, such that $A = \nu(H)$, $\forall i \in [1, n] \nu(L_i)$ is satisfiable, and $\nu(c_1, \ldots, c_m)$ satisfiable. Following the Lemma 1, ν is also a valuation based on $\mathcal{D}^{ext}_{\mathcal{I}_2}$. Since $\mathcal{I}_1 \subseteq \mathcal{I}_2$, we have $\nu(L_i)$ satisfiable $\forall i \in [1, n]$, and $\nu(c_1, \ldots, c_m)$ satisfiable. Hence $A \in T_P(\mathcal{I}_2)$.

■

Theorem 1 (Continuity) *The operator T_P is continuous, that is, if $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \ldots$ are interpretations such that $\mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \mathcal{I}_3 \ldots$ (possibly infinite sequence), then $T_P(\bigcup_i \mathcal{I}_i) \subseteq \bigcup_i T_P(\mathcal{I}_i)$.*

Proof Let $\mathcal{I} = \bigcup_i \mathcal{I}_i$ and let A be an atom in $T_P(\mathcal{I})$. We must show that A is also in $\bigcup_i T_P(\mathcal{I}_i)$. At least one of the following two cases applies:

- $A \in \mathcal{I}$, i.e., $A \in \bigcup_i \mathcal{I}_i$. Then, there exists some j such that $A \in \mathcal{I}_j$. Thus, $A \in T_P(\mathcal{I}_j)$ and consequently $A \in \bigcup_i T_P(\mathcal{I}_i)$.
- There exists a rule $r : H \leftarrow L_1, \ldots, L_n \| c_1, \ldots, c_m$ in P and a valuation ν based on $\mathcal{D}^{ext}_{\mathcal{I}}$ such that $\forall i \in [1, n]$ $\nu(L_i)$ is satisfiable and $\nu(c_1, \ldots, c_m)$ satisfiable. Since $\nu(L_i)$ satisfiable, there exists some j_i such that $\nu(L_i)$ satisfiable in $\mathcal{I}_{j_i}$. In addition, since

the $\mathcal{I}_k$ are increasing, there exists some l, such that $\mathcal{I}_{j_i} \subseteq \mathcal{I}_l$ for all j_i. Hence, $\nu(L_i)$ satisfiable in $\mathcal{I}_l$ $\forall i \in [1,n]$ and $\nu(c_1,\ldots,c_m)$ satisfiable. Let $V = var(L_1,\ldots,L_n)$ be the set of variables in the rule r, and let $\nu(V)$ be the result of applying ν to each variable in V. $\nu(V)$ is a finite subset of $\mathcal{D}_{\mathcal{I}}^{ext}$ since ν is based on $\mathcal{D}_{\mathcal{I}}^{ext}$. We have $\nu(L_i)$ satisfiable $\forall i \in [1,n]$ and $\nu(c_1,\ldots,c_m)$ satisfiable. Thus, $\nu(var(L_i))$ satisfiable in $\mathcal{D}_{\mathcal{I}_l}^{ext}$ $\forall i \in [1,n]$ and $\nu(c_1,\ldots,c_m)$ satisfiable. Then $A \in T_P(\mathcal{I}_l)$ $(A = \nu(H))$. Consequently $A \in \bigcup_i T_P(\mathcal{I}_i)$.

■

Lemma 3 *$\mathcal{I}$ is a model of P iff $T_P(\mathcal{I}) \subseteq \mathcal{I}$.*

Proof
" $\Rightarrow$ " If $\mathcal{I}$ is an interpretation and P a program, then let $cons(P,\mathcal{I})$ denote the set of all ground facts which are immediate consequences for $\mathcal{I}$ and P.

$$T_P(\mathcal{I}) = \mathcal{I} \cup \{A \mid A \text{ is an immediate consequence for } \mathcal{I} \text{ and } P\}$$

For any element A in $cons(P,\mathcal{I})$, at least one of the following cases holds:

- $A \in \mathcal{I}$. By definition of immediate consequence;
- there exists a rule $r : H \leftarrow L_1,\ldots,L_n \| c_1,\ldots,c_m$ in P, and a valuation ν such that $\forall i \in [1,n]$ $\nu(L_i)$ satisfiable in $\mathcal{I}$, $\nu(c_1,\ldots,c_m)$ satisfiable, and $A = \nu(H)$. Since $\mathcal{I}$ is a model of P, $\mathcal{I}$ satisfies r $(\mathcal{I} \models r)$, and then $A \in \mathcal{I}$. Thus, $T_P(\mathcal{I}) \subseteq \mathcal{I}$.

" $\Leftarrow$ " Let $\mathcal{I}$ be an interpretation and P be a program.
Let $r : H \leftarrow L_1,\ldots,L_n \| c_1,\ldots,c_m$ be any rule in P and ν any valuation. If $\forall i \in [1,n] \nu(L_i)$ satisfiable in $\mathcal{I}$ and $\nu(c_1,\ldots,c_m)$ satisfiable in $\mathcal{I}$, then $\nu(H) \in T_P(\mathcal{I})$. Because $T_P(\mathcal{I}) \subseteq \mathcal{I}$, we have $\nu(H) \in \mathcal{I}$, and then $\mathcal{I}$ satisfies r $(\mathcal{I} \models r)$. Hence $\mathcal{I} \models P$. ■

Lemma 4 *Each fixpoint of T_P is a model for P.*

Proof Follows immediately from Lemma 3. ■

Theorem 2 *Let P be a program and $\mathcal{I}$ an input such that the minimal model for P exists, then the minimal model and the least fixpoint coincide.*

Proof Let P be a program and $\mathcal{I}$ an interpretation, Let us denote by $P(\mathcal{I})$ the minimal model of P containing $\mathcal{I}$. According to lemma 3, $T_P(P(\mathcal{I})) \subseteq P(\mathcal{I})$. T_P is monotonic, so $T_P(T_P(P(\mathcal{I}))) \subseteq T_P(P(\mathcal{I}))$, and then $T_P(P(\mathcal{I}))$ is a model of P containing $\mathcal{I}$. As $P(\mathcal{I})$ is the minimal model containing $\mathcal{I}$, we have $P(\mathcal{I}) \subseteq T_P(P(\mathcal{I}))$. As $P(I)$ is a fixpoint of P and also a minimal model of P, each fixpoint of T_P containing $\mathcal{I}$ is a model of P containing $P(\mathcal{I})$. Thus $P(\mathcal{I})$ is the minimal model of P containing $\mathcal{I}$. ■

7 Query Containment

Containment[7] and *equivalence* of queries is the problem of checking whether the result of one query is contained in or equal to, what another query produces [2, 67]. Containment is mainly concerned with query optimization. For example, in the case of the use of materialized views for rewriting queries, a given query is compared against multiple view definitions (which may be seen as queries as well) to identify which of them may be combined to answer the query. This comparison is essentially testing for query containment.

In this section we present an algorithm for deciding the containment of a query within a view (which is a query as well). Our method is inspired by the one given in [68].

Definition 13 *(Containment) Given a query Q and a view V in our query language, are the answers to Q also answers to V in any database state.*

The following algorithm takes as input a query $Q\|C$ and a view $V\|C'$ and checks whether the query is contained in the view. We call this algorithm **QCD** (Query Containment Decision).

Algorithm QCD
(Checking Containment of a query in a view)

Require: a query $Q\|C$ and a view $V\|C'$
Ensure: $Q\|C$ is contained in $V\|C'$ or not.
1: Generate a canonical database $\mathsf{C_aDB}$ form $Q\|C$
   // $\mathsf{C_aDB} = H_Q \cup B_Q \cup C_Q$
   // H_Q: the instanciated head of the query $Q\|C$
   // B_Q: the instanciated body of the query $Q\|C$
   // C_Q: the instanciated constraint part of the query $Q\|C$
2: Evaluate the view $V\|C'$ over $\mathsf{C_aDB}$. Let $\mathsf{F_{C_aDB}}$ be the result (a set of facts).
3: If $\mathsf{F_{C_aDB}} \subseteq H_Q$
4: then return $Q\|C$ contained in $V\|C'$
5: else return $Q\|C$ not contained in $V\|C'$

In the following, we explain the different steps of the algorithm.

Intuitively, the QCD algorithm discovers the containment by "matching" the view with subgoals of the query. In particular, the matching is done as follows: First we create a prototypical database containing "frozen fact" for every subgoal of the query. Frozen facts are derived by turning the object variables in the query into unique constants which will be denoted by a bar.

Example 1 *Consider the following query:*

$$\mathsf{Answer(X,Y)} : -\mathsf{link(X,Y,}\alpha\mathsf{)}\|\beta\dot{\prec}\alpha, \beta\dot{\in}\mathsf{video.sequences}$$

[7] Also called implication.

which consists in retrieving a set of pairs (X, Y) *such that* Y *is reachable from* X *by a path starting with the sub-path* video.sequences *and ending with any sub-path.*

The prototypical database for this query is then:

$$\mathsf{Answer}(\bar{x},\bar{y}), \mathsf{link}(\bar{x},\bar{y},\alpha), \beta \dot{\prec} \alpha, \beta \dot{\in} \mathsf{video.sequences}$$

We distinguish three parts in this database:

- H_Q which is the singleton containing the frozen fact derived from the head of the query Q, i.e., $H_Q = \{\mathsf{Answer}(\bar{x},\bar{y})\}$.
- B_Q which is the set of frozen facts derived from the body of the query, i.e., $B_Q = \{\mathsf{link}(\bar{x},\bar{y},\alpha)\}$.
- C_Q the set of constraints in Q after object variables are turned into constants, i.e., $C_Q = \{\beta \dot{\prec} \alpha, \beta \dot{\in} \mathsf{video.sequences}\}$.

The QCD algorithm then evaluates the view over a subset of the prototypical database (here $B_Q \cup C_Q$), trying to derive the frozen fact corresponding to the head of the query (i.e., the content of H_Q).

Example 2 *Consider the following view:*

$$\mathsf{Answer}(\mathsf{U},\mathsf{V}) : -\mathsf{link}(\mathsf{U},\mathsf{V},\gamma) \| \rho \dot{\prec} \gamma, \rho \dot{\in} \mathsf{video}$$

the evaluation of the body of this view over $B_Q \cup C_Q$ leads to the following ground instance of the view

$$\mathsf{Answer}(\bar{x},\bar{y}) : -\mathsf{link}(\bar{x},\bar{y},\alpha) \| \beta \dot{\prec} \alpha, \beta \dot{\in} \mathsf{video}, \beta \dot{\prec} \alpha, \beta \dot{\in} \mathsf{video.sequences}$$

with the substitution $\sigma = \{U/\bar{x}, V/\bar{y}, \gamma/\alpha, \rho/\beta\}$. Note that the constraint part of the prototypical database (i.e., C_Q) is added to the constraint part of the view. In our case, the body and the constraint parts of the instanciated view are satisfiable and then we can derive the head of the view which is contained in H_Q.

Theorem 3 *(Correctness) Algorithm QCD terminates and produces* yes *is the query entails the view and* no *otherwise.*

8 Related Work

We discuss the relationship of our work to *query languages for semistructured data* and *path queries with constraints.* This section is intended to be illustrative. We apologize if we left out other relevant works.

Query languages for semistructured data. Semistructured data is modeled as labeled graph, in which the nodes correspond to the objects and the edges to their attributes. Most query languages proposed for semistructured data can navigate the data using *Regular Path Expressions*, thus traversing arbitrary long paths in the graph.

In [3], for example, a query language, called Lorel, for semistructured data is obtained by extending OQL[20] with powerful and flexible path expressions, which allow querying without precise knowledge of the structure. Path expressions are built from labels and wild-cards (place-holders) using regular expressions, allowing the user to specify rich patterns that are matched to actual paths in the database graph. One of the limits of this language is that it does not allow to express recursive queries over database graphs.

The language reported in [17], UnQL, is closely related to Lorel, allowing to query data organized as a root, edge-labeled graph. A primary feature of UnQL is a powerful construct called *traverse* that allows restructuring of trees to arbitrary depth. The language of terms uses variables ranging over trees or over edge labels. A tree is seen as a set of edge/subtree pairs. For example, in the expression of the form $\backslash l \Rightarrow \backslash t \leftarrow DB$, the label variable $\backslash l$ is used to match any edge emanating from the root of the database DB. the variable $\backslash t$ will be bound to the associated subtree. Certain restructuring queries, which require a fixpoint operation, appear not to be expressible in UnCAL, a calculus for UnQL.

[47] proposed a SQL-like query language (called WebSQL) that integrates textual retrieval with structure and topology-based queries. The language is designed to query the World Wide Web. To make reference to the hypertext structure of the web, the language uses a set of symbols allowing to define path regular expressions. For example, $=|\Rightarrow \,.\, \rightarrow^*$ is a regular expression that represents the set of paths containing the zero length path and all paths that start with a global link and continue with zero or more local links. A hypertext link is said to be local if the destination and the source documents are different but located on the same server, and it is said to be global if the destination and the source documents are located on different servers. In this language, queries may contain constraints like xvy, where v is a variable ranging over paths. Again, this proposal does not allow to express recursive queries which may be extremely useful when querying the Web.

In [22, 23], extensions to OQL are proposed that are somewhat similar in spirit or goals to LOREL. In [22], a more rigidly typed approach is followed, but because heterogeneous collections are introduced, the model still has a strong similarity to OEM. However, the language proposed in [22], called OQL-doc, does not use coercion the way it is used in LOREL, and the treatment of path expressions is quite different. Optimizing the evaluation of *generalized path expressions* is considered in [23]. Their optimization is based on two object algebra operators, one dealing with paths at the schema level and one with paths at the data level.

[33] investigated conjunctive queries that allow for incomplete answers in the framework of semistructured data. The proposed model of query evaluation consists of a search phase

(involving search constraints), where a query graph containing variables is used to match a maximal portion of the database graph, and a filter phase (involving filter constraints) where the maximal matchings resulting from the search phase are subjected to constraints. The authors deliberately limited their investigation to queries that do not allow regular path expressions.

Also related to our work are several queries for the World-Wide Web that have emerged recently, e.g., W3QL [39], which focuses on extensibility, and *WebLog* [41] which is based on a Datalog-like syntax. Additional relevant work includes query languages for hypertext structures, e.g., [13, 26, 49, 46], and work on integrating SGML [29] documents with relational databases [14], since SGML documents can be viewed as semistructured data.

In the area of heterogeneous database integration, which is a common scenario for semistructured data, most of work has focused on integrating data in well structured databases. In particular, systems such as Pegasus [55] and UniSQL/M [38] are designed to integrate data in object-oriented and relational databases. At the other end of the spectrum, systems such as GAIA [56] and ACL/KIF [28] provide uniform access to data with minimal structure.

[39] also incorporated path expressions. In the proposed language, the condition part of a query may contain expressions of the form $path = regexp$ where path is a path expression and $regexp$ is a Perl [59] regular expression.

path queries with constraints. Abiteboul and Vianu [5] investigated the evaluation and optimization of path expression queries involving path constraints. Path constraints are local; they may capture the structural information about a web site (or a collection of sites). A path constraint is an expression of the form $p \subseteq q$ or $p = q$ where p and q are regular expressions. A path constraint $p \subseteq q$ holds at a given site if the answer to query p applied to that site is included in the answer to q applied to the same site (and similarly to $p = q$). The constraint $p = q$ is also allowed in our constraint language. Constraints such as $\alpha \dot{\prec} \beta$ and $\alpha \dot{\amalg} \beta$ cannot be expressed as word constraints [5].

Buneman *et al.* [18] proposed a class of path constraints that are useful for both structured and semistructured data for specifying natural integrity constraints. A path constraint φ is an expression of either the forward form $\forall xy(p(r,x) \wedge q(x,y) \rightarrow \gamma(x,y))$ or the backward form $\forall xy(p(r,x) \wedge q(x,y) \rightarrow \gamma(y,x))$ where p, q, γ are paths. This constraint language cannot express queries like $\alpha \dot{\prec} \beta$ and $\alpha \dot{\amalg} \beta$.

9 Conclusion

There is a growing interest in semistructured databases. As such data (e.g., on the Web) proliferate, aids to browsing and filtering become increasingly important tools for interacting with such exponentially growing information resources and for dealing with access problems.

In this paper, we have presented a class of path constraints and addressed the problem of developing a formal, rule-based constraint query language that allows the retrieval of semistructured data. The primary motivation of this work was that path constraints are relevant in semistructured data retrieval and the absence of suitable support for expressing such constraints in traditional query languages represent a serious obstacle.

Our approach is purely declarative and formulated in terms of constraints between path variables which straightforwardly capture, for example, what it means for two paths to be divergent. The implementation of query evaluation procedures requires efficient algorithms for solving path constraints. A formal account of constraint languages for semistructured data is an essential step in demonstrating the correctness of such algorithms, and may yield more efficient processing strategies.

References

[1] S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of the International Conference on Database Theory (ICDT'97), Delphi, Greece*, pages 1–18, Janvier 1997.

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

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] S. Abiteboul and V. Vianu. Queries and Computation on the Web. In F. N. Afrati and P. Kolaitis, editors, *In Proceedings of the 6th International Conference on Database Theory (ICDT'97), Delphi, Greece*, volume 1186 of *Lecture Notes in Computer Science*, pages 662–675. Springer, 1997.

[5] S. Abiteboul and V. Vianu. Regular Path Queries with Constraints. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databases (PODS'97), Tucson, Arizona*, pages 122–133. ACM Press, May 1997.

[6] H. Aït-Kaci. An Algebraic Semantics Approach to the Effective Resolution of Type Equations. *Theoretical Computer Science*, 45:293–351, 1986.

[7] H. Aït-Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-in Inheritance. *Journal of Logic Programming*, 3(3):185–215, 1986.

[8] H. Aït-Kaci and A. Podelski. Towards a Meaning of LIFE. *The Journal of Logic Programming*, 16(3-4), July 1993.

[9] H. Aït-Kaci, A. Podelski, and G. Smolka. A Feature-Based Constraint System for Logic Programming with Entailment. *Theoretical Computer Science*, 122:263–283, 1994.

[10] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and Z. Zdonik. The Object-Oriented Database Manifesto. In *In Proceedings of the First International Conference on Deductive and Object Oriented Databases (DOOD'89), Kyoto, Japan*, pages 40–57, 1989.

[11] F. Baader, H. J. Bückert, B. Nebel, W. Nutt, and G. Smolka. On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.

[12] R. Backofen. Regular Path Expressions in Feature Logic. *Journal of Symbolic Computation*, 17:421–455, 1994.

[13] C. Beeri and Y. Kornatski. A Logic Query Language for Hypermedia Systems. *Information Systems*, 77, 1994.

[14] G. Blake, M. Consens, P. Kilpeläinen, P. Larson, T. Snider, and F. Tompa. Text/relational Database Management Systems: Harmonizing SQL and SGML. In *Proceedings of the First International Conference on Applications of Databases, Vadstena, Sweden*, pages 267–280, 1994.

[15] R. J. Brachman and H. J. Levesque. The Tractability of Subsumption in Frame-based Description Languages. In *Proceedings of the National Conference on Artificial Intelligence*, pages 34–37, Aug. 1984.

[16] P. Buneman. Semistructured Data. In *Proceedings of the ACM Symposium on Principles of Database Systems, Tucson, Az, USA*, 1997.

[17] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM SIGMOD International Conference (SIGMOD'96), Montreal, Canada*, pages 505–516, June 1996.

[18] P. Buneman, W. Fan, and S. Weinstein. Path Constraints on Semistructured and Structured Data. In *Proceedings of the seventeenth ACM-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98), Seattle, Washington*, pages 129–138. ACM Press, 1998.

[19] H.-J. Bürckert. A Resolution Principle for Constrained Logics. *Artificial Intelligence*, 66:235–271, 1994.

[20] R. G. G. Cattell. The Object Database Standard: ODMG-93. Maurgan Kaufmann, San Francisco, California. 1994.

[21] C. Chang, H. Garcia-Molina, and A. Paepcke. Boolean Query Mapping Across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4), 1996.

[22] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'94)*, pages 313–324, May 1994.

[23] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pages 313–322, June 1996.

[24] E. F. Codd. Extending the Database Relational Model to Capture more Meaning. *ACM Transactions on Database Systems*, 4:397–434, 1979.

[25] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proceedings of the 2nd International Conference on 5th Generation Computer Systems*, pages 85–99, 1984.

[26] M. P. Consens and A. O. Mendelzon. Expressing Structural Hypertext Queries in Graphlog. In *Proceedings of the Second ACM Conference on Hypertext, Pittsburgh, Pennsylvania*, Nov. 1989.

[27] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for a Web Site Management System. *SIGMOD Record*, 26(3):4–11, 1997.

[28] M. Genesereth and R. Fikes. Knowledge Interchange Format Reference Manual. Available as http://logic.stanford.edu/sharing/papers/kif.ps. 1994.

[29] C. F. Goldfarb and Y. Rubinski. The SGML Handbook. *Clarendon Press, Oxford, UK*, 1990.

[30] M.-S. Hacid, C. Decleir, and J. Kouloumdjian. A Database Approach for Modeling and Querying Video Data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(5):729–750, 2000.

[31] M. Höhfeld and G. Smolka. Definite Relations over Constraint Languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, Oct. 1988.

[32] M. Johnson. Attribute-Value Logic and the Theory of Grammar. CSLI Lectures Notes 16, Center for the Study of Language and Information, 1988.

[33] Y. Kanza, W. Nutt, and Y. Sagiv. Queries with Incomplete Answers over Semistructured Data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databasests (PODS'99), Philadelphia, Pennsylvania*, pages 227–236. ACM Press, May 1999.

[34] R. M. Kaplan and J. Bresnan. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan, editor, *The mental Representation of Grammatical Relations*, pages 173–381. MIT Press, Cambridge (MA), 1982.

[35] R. T. Kasper and W. C. Rounds. A Logical Semantics for Feature Structures. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics*, pages 257–265, 1986.

[36] M. Kay. Functional Grammar. In C. Chiarello, editor, *Proceedings of of the fifth Annual Meeting of the Berkeley Linguistics Society*, pages 142–158, 1979.

[37] M. Kay. Parsing in Functional Unification Grammar. In *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives, Chapter 7*, pages 251–278. Cambridge University Press, 1985.

[38] W. Kim. On Object Oriented Database Technology. UniSQL Product Literature. 1994.

[39] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proceedings of the 21th International Conference on Very Large Databases (VLDB'95), Zurich, Switzerland*, pages 54–65. Morgan Kaufmann, Sept. 1995.

[40] G. Kuper, L. Libkin, and J. Paradaens. *Constraint Databases*. Springer-Verlag, March 2000.

[41] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proceedings of the Sixth International Workshop on Research Issues in Data Engineering (RIDE'96)*, Feb. 1996.

[42] J. W. Lioyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. Second edition.

[43] K. Marriott and P. J. Stuckey. *Programming With Constraints : An Introduction*. MIT Press, January 1999.

[44] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. LORE: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[45] G. Mecca and A. J. Bonner. Sequences, Datalog and Transducers. In *Proceedings of the 1995 Symposium on Principles of Database Systems (PODS'95), San Jose, California*, pages 23–35, May 1995.

[46] A. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal of Computing*, 24(6), 1995.

[47] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54–67, 1996.

[48] A. O. Mendelzon and T. Milo. Formal Models of Web Queries. In *Proceedings of the Sixteenth ACM-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97), Tucston, Arizona*, pages 134–143. ACM Press, 1997.

[49] T. Minohara and R. Watanabe. Queries on Structure in Hypertext. In *Foundations of Data Organization and Algorithms (FODO'93)*, pages 394–411, 1993.

[50] K. Mukai. Partially Specified Terms in Logic Programming for Linguistic Analysis. In *Proceedings of the 6th International Conference on Fifth Generation Computer Systems, Tokyo, Japan*, 1988.

[51] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* LNCS-422. Springer-Verlag, 1990.

[52] B. Nebel and G. Smolka. Representation and Reasoning with Attribute Descriptions. In K. H. Bläsius, U. Hedtstück, and C.-R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, volume 418 of *LNAI*, pages 112–139. Springer Verlag, 1990.

[53] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In L. M. Haas and A. Tiwary, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 295–306, Seattle, Washington, USA, June 1998. ACM Press.

[54] C. Pollard and I. Sag. *Head Driven Phrase Structure Grammar. Studies in Contemporary Linguistics.* Cambridge University Press, Cambridge, England, 1994.

[55] A. Rafii, R. Ahmed, M. Ketabchi, P. DeSmedt, and W. Du. Integrating Strategies in the Pegasus Object Oriented Multidatabase System. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume II, pages 323–334, 1992.

[56] R. Rao, B. Janssen, and A. Rajaraman. GAIA Technical Overview. Technical Report, Xerox Palo Alto Research Center. 1994.

[57] J. A. Robinson. A Machine Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1), 1965.

[58] W. C. Rounds. Feature Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 475–533. Elsevier Science Publishers B.V. (North Holland), 1997. Part 2: General Topics.

[59] R. L. Schwartz. *Learning Perl.* O'Reilly & Associates, Inc., 1993. Ch. Regular expressions.

[60] S. Shieber, H. Uszkoreit, F. Pereira, J. A. Robinson, and M. Tyson. The Formalism and Implementation of PART-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge.* SRI International, Menlo Park, California, 1983.

[61] S. M. Shieber. The Design of a Computer Language for Linguistic Information. In *Proceedings of COLING'84*, pages 363–366. 1984.

[62] S. M. Shieber. An Introduction to Unification-Based Approaches to Grammar. *volume 4 of CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University*, 1986.

[63] G. Smolka. Feature Constraint Logics for Unification Grammars. *Journal of Logic Programming*, 12(1-2):51–87, 1992.

[64] G. Smolka. The Oz programming model. In J. van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin, 1995.

[65] G. Smolka and R. Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, Apr. 1994.

[66] J. Thierry-Mieg and R. Durbin. Syntactic Definitions for the ACeDB Data Base Manager. Technical report mrc-lmb, MRC Laboratory for Molecular Biology, 1992.

[67] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I, II. Computer Science Press, Rockville MD, 1989.

[68] V. Vassalos and Y. Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97), Athens, Greece*, pages 256–265, 1997.

[69] J. W. Y. Papakonstantinou, H. Garcia Molina. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95), Taipei, Taiwan*, pages 251–260, Mars 1995.