Ordering and Path Constraints over Semistructured Data*

Elisa Bertino* **Ahmed K. Elmagarmid†** **Mohand-Saïd Hacid‡**

*Dipartimento di Scienze dell'Informazione
University of Milano
Via Comelico, 39/41 20135 Milani - Italy
bertino@dsi.unimi.it

†Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 - USA
ake@cs.purdue.edu

‡Laboratoire d'Ingénierie des Systèmes d'Information
20, avenue Albert Einstein
69621 Villeurbanne - France
mshacid@lisi.insa-lyon.fr

Abstract. *Constraints are a valuable tool for managing information. Feature constraints have been used for describing records in constraint programming [8, 54] and record like structures in computational linguistics [35, 52]. In this paper, we consider how constraint-based technology can be used to query and reason about semistructured data. The constraint system $FT_{\leq}$ [46] provides information ordering constraints interpreted over feature trees. Here, we show how a generalization of $FT_{\leq}$ combined with path constraints can lead to formally represent, state constraints, and reason about semistructured data. The constraint languages we propose provide possibilities to straightforwardly capture, for example, what it means for a tree to be a subtree or subsumed by another, or what it means for two paths to be divergent. We establish a logical semantics for our constraints thanks to axiom schemes presenting our first-order theory constraint system. We propose using the constraint systems for querying semistructured data.*

Keywords: Semistructured Data, Constraints, Role Tress, Semantics, Satisfiability, Rule Languages.

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, 44], biological databases [55], 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).

*This work is supported by National science Foundation under grants 9972883-EIA and 9983249 EIA, and a grant from HP.

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 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.

Recent research works propose to model semistructured data using "*lightweight*" data models based on labeled directed graphs [1, 17]. Based on this data structure, 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 Feature Logics[1], instead of algebraic programming. In particular, we develop a rule-based constraint query language for manipulating semistructured data. The resulting language has both a clear declarative semantics and an operational semantics. The semantics is based on fixpoint theory [40]. Relevant features of the proposed language is the support for recursive queries, order constraints and path constraints, which cannot be expressed in other languages for semistructured data. Those constraints support a wide range of query predicates, such as inclusion among data tree structures, compatibility between trees, or divergence between paths.

Paper outline: Section 2 summarizes the contributions of this paper. Section 3 introduces semistructured data. In Section 4 we define the data model and give examples of queries. Section 5 presents the syntax and semantics of two new constraint languages suitable for semistructured data. In Section 6, we develop our query language and give its syntax and semantics. Section 7 discusses constraints relaxation in order to compute approximate query answers. Section 8 deals with the connection of the proposed framework with XML. Section 9 discusses related work. We conclude in Section 10.

2 Contributions

Semistructured databases have reached a widespread acceptance in practical applications of databases (see XML). They are acknowledged for their simplicity combined with their expressibility. As such, the field of semistructured databases has proven to be an important research platform and in many ways setting the standard for future database technology. In order to meet the demands for more expressive and flexible ways to interact with semistructured database, it is important to go beyond

[1]Feature logic (see, e.g., [11, 53, 33, 36]) has its origin in the three areas of knowledge representation with *concept descriptions*, *frames*, or Ψ-*terms* [6], natural language processing, in particular approaches based on *unification grammars* (see, e.g., [50]), and constraint (logic) programming (see, e.g., [7, 8]).

what can be formalized by traditional tools. In this paper, we introduce a new constraint system for semistructured data. The constraint system consists of two interacting constraint languages. The resulting query language gives the user the ability to define a broad class of queries that cannot be (naturally) expressed by means of existing formal languages for semistructured data.

We present two classes of constraints, namely, ordering constraints and path constraints, that are of interest in connection with both structured and semistructured databases. Our constraints are inspired by Feature Logics. 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 [52]. Feature descriptions have also been proposed as a constraint system for logic programming (see, for example, [9, 54]). They provide for a partial description of abstract objects by means of functional attributes, called features. On top of our constraint languages we allow the definition of relations (by means of definite clauses) in the style of [32], leading to a 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. The satisfiability for conjunctions of constraints is decidable. Thus, an unsatisfiable query denotes the empty set in every interpretation, which means that it is worthless.

To summarize, the framework presented here integrates formalisms developed in Databases, Feature Logics and Constraint (Logic) Programming. The paper builds on the works by [9, 12, 46, 19] to propose a new constraint system for semistructured data and a declarative, rule-based, constraint query language that has a clear declarative and operational semantics. We make the following contributions:

(1) We develop a simple and flexible structure for representing semistructured data. The structure, called *role trees*, is inspired by Feature Constraint Systems. Trees are useful for structuring data in modern applications. This gives the more flexible role trees (our data structure) an interesting potential.

(2) We propose two constraint languages for semistructured data. The ordering constraints allow to declaratively specify relationships between trees representing semistructured data. Path constraints allow to constrain the navigation of the trees. Our constraints are of a finer grain and of different expressiveness.

(3) We propose a declarative, rule-based, constraint query language that can be used to infer relationships between semistructured data. We view our query language as consisting of two constraint languages on top of which relations can be defined by definite clauses. The language has declarative and operational semantics. The semantics is based on fixpoint theory, as in classical logic programming, and on a new notion of active domain, called the *extended active domain*, introduced to allow for a bottom-up evaluation of rules.

(4) Approximate answers make it necessary to refine the model of query evaluation. For that, we propose two rewriting rules allowing to relax some constraints in queries.

As usual in information-intensive applications (e.g., databases), a *declarative* specification of constraints and queries should be preferred to more *procedural* one: it is usually more concise and elegant

because it is likely to support formal analysis and thence optimization by the DBMS.

To our knowledge (see related work in Section 9), no previous work considers the kind of constraints we propose and their use in the context of semistructured data.

3 Semistructured Data

In traditional databases such as the relational model [25] 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 [55], 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 information, 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, eXtensible 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 [41], a repository for OEM [56] 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.

4 Data and Query Modeling

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[2] of a semistructured database is given in 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 paths along the graph, and can be viewed as compositions of labels. For example, the expression

video.category

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

The Graph-Oriented Model. Formally, semistructured data is represented by a directed labeled graph $G = (N, E)$ where N is a finite set of labeled nodes and E is a finite set of labeled edges. An edge e is written as (n_1, α, n_2) where n_1 and n_2 are members of N and α is the label of the edge. The label of a node n is given by a function $\lambda(n)$ that maps to a non-null string. The label α of an edge $e = (n_1, \alpha, n_2)$ is a string given by $\alpha = \sigma(e)$. The domain of the functions λ and σ is the universal set of all nodes (from all graphs) and the range is the set of strings (from all lexicons).

Our constraint system avoids overspecification by allowing the description

x : conference[name ⇒ {VLDB}, location ⇒ {Roma},
program_co_chairs ⇒ {Snodgrass, Apers, Ramamohanarao}]

[2]This example is inspired by the one given in [31].

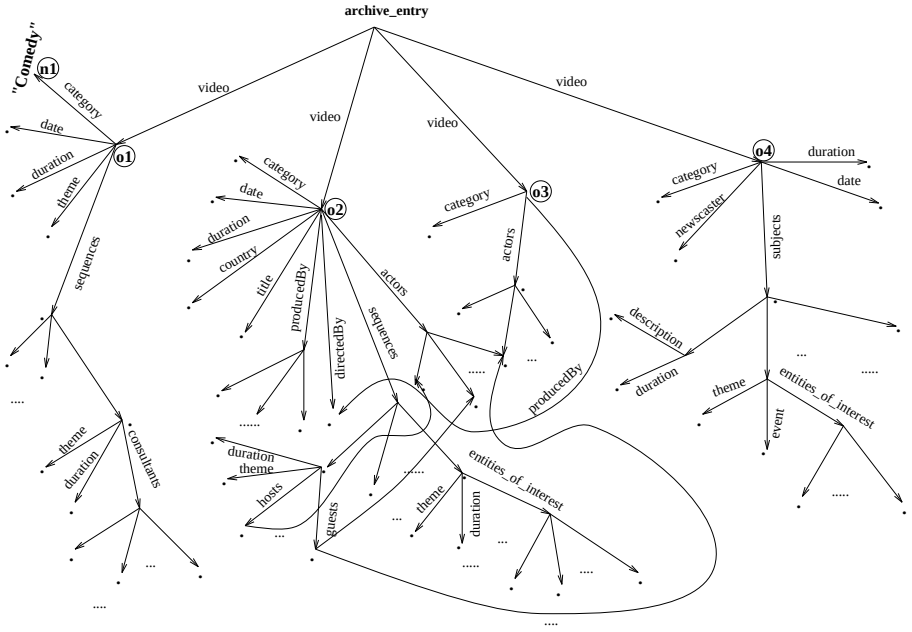

Figure 1: A video database content.

saying that x has sort conference, its role *name* is VLDB, its role *location* is Roma, and its role *program_co_chairs* is Snodgrass, Apers, and Ramamohanarao. Nothing is said about other roles of x, which may or may not exist.

In this paper, we use the notion of trees to represent semistructured data. We investigate a set of constraints over semistructured data. Before presenting these constraint languages, we shortly and informally discuss feature and role trees.

A *feature tree* is a tree with unordered, labeled edges and labeled nodes. The edge labels are called features; features are functional in that two features labeling edges departing from the same node are distinct. In programming, features correspond to record field selectors and node labels to record field contents. In our framework, we extend the notion of feature trees to role trees. A role tree is a possibly infinite tree whose nodes are labeled with symbols called sorts, and whose edges are labeled with symbols called roles. The labeling with roles is nondeterministic in that edges departing from a node need not to be labeled with distinct roles.

An example of a role tree is shown in figure 1. Its root is labeled with the node label archive_entry and the edges departing at this root are labeled by the role video.

A role tree is defined by a tree domain and a labeling function. The domain of a role tree τ is the multiset of all words labeling a branch from the root of τ to a node of τ. For instance, the domain of the tree of figure 2 is $\{\epsilon$, video, video, video.category, video.date, video.duration, video.producedBy, etc.$\}$. The labeling function associates each element of the domain with a set of sorts.

A role tree is finite if its tree domain is finite. In general, the domain of a role tree may also be infinite in order to model semistructured data with cyclic dependencies.

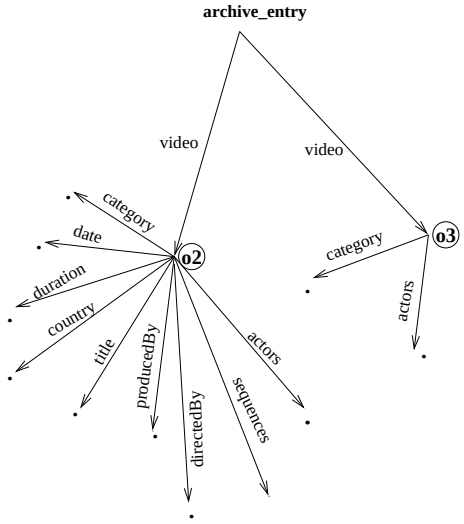

Figure 2: Example of tree.

A role tree can be seen as a carrier of information. This viewpoint gives rise to an ordering relation on role trees in a very natural way that we call *information ordering*. The information ordering is illustrated by the example of figure 3. The smaller tree is missing some information about the object it represents, namely that this object is an archive video and that role *category* of the object $\mathbf{O_1}$ is *Western* and the role *actor* is *Steve McQueen* and *Youn Bruner*. In order to have nodes without information, we allow for unlabeled nodes depicted with a $\bullet$. Formally, this means that we do not require a labeling function to be total.

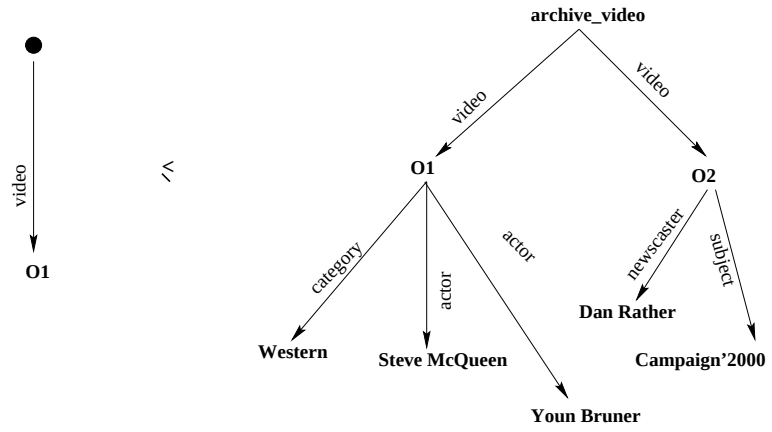

Figure 3: Example of an order over trees.

Intuitively, a role tree τ_1 is smaller than a role tree τ_2 if τ_1 has fewer edges and node labels than τ_2. More precisely, this means that every word of roles in the tree domain of τ_1 belongs to the tree domain of τ_2 and that the partial labeling function of τ_1 is contained in the labeling function of τ_2. In this case we write $\tau_1 \leq \tau_2$. The notions of tree domains and labeling function will be formally defined later.

The following are examples of queries (over databases of the style of figure 1). In these queries, x and y are tree variables (i.e., ranging over trees), and α, β are path variables (i.e., ranging over composition of roles). We use the predicate symbol tree to denote the set of trees in the database. The formal

semantics of the constructs used in constraints will be given later.

$$\mathsf{answer(x)} \leftarrow \mathsf{tree(x)} \| \{\mathsf{Deniro}, \mathsf{Devito}\}(\alpha, \mathsf{x}), \alpha \dot{\in} \mathsf{video.actor}$$

This query returns the set of trees such that there is a path video.actor from the root (of each tree answer to the query), leading to the set of sorts {Deniro, Devito}. The symbol $\dot{\in}$ is used to express path restriction. In this query (which is expressed as a rule), answer(x) is called the head of the query, tree(x) is called the body of the query, and $\{\mathsf{Deniro}, \mathsf{Devito}\}(\alpha, \mathsf{x}), \alpha \dot{\in} \mathsf{video.actor}$ is called the constraint part of the query. The notation $\mathsf{S}(\alpha, \mathsf{x})$ means that in the tree x there is a path α from the root to the set of sorts S. For example, on the right of figure 3, the path video.category leads to the singleton {Western}.

$$\mathsf{answer(x, y)} \leftarrow \mathsf{tree(x)}, \mathsf{tree(y)} \| \mathsf{x} \leq \mathsf{y}, \{\mathsf{Tom\_Hanks}\}(\alpha, \mathsf{x}), \{\mathsf{Tom\_Hanks}\}(\beta, \mathsf{y}), \alpha \dot{\amalg} \beta$$

This query returns a set of pairs (x, y) of trees such that there is a path α in x and a path β in y, that are divergent (i.e., different) and leading to the same set of sorts (here {Tom_Hanks}). The symbol $\dot{\amalg}$ stands for divergence of paths.

$$\mathsf{answer(x, y)} \leftarrow \mathsf{tree(x)}, \mathsf{tree(y)} \| \mathsf{x} \sim \mathsf{y}$$

This query returns pairs of trees that are compatible. The symbol $\sim$ stands for compatibility. Two trees x and y are compatible if there is another three z such that x and y are both subsumed by z. In other words, $x \sim y \Leftrightarrow \exists z(x \leq z \wedge y \leq z)$.

5 Constraint Languages for Semistructured Data

This section introduces the core aspects of our model for semistructured data. We first formally define the notion of trees. We then present the two most relevant features of our model, namely ordering constraints and path constraints. We also discuss issues related to satisfiability and expressiveness of those constraints.

5.1 Role Trees and Constraints

To give a rigorous formalization of role trees, we first fix two disjoint alphabets $\mathcal{S}$ and $\mathcal{F}$, whose symbols are called *sorts* and *roles*, respectively. The letters S, S' will always denote sets of sorts, and the letters f, g will always denote roles. Words over $\mathcal{F}$ are called paths. The concatenation of two paths v and w results in the path vw. The symbol ϵ denotes the empty path, $v\epsilon = \epsilon v = v$, and $\mathcal{F}^*$ denotes the set of all paths.

A *tree domain* is a nonempty set $D \subseteq \mathcal{F}^*$ that is prefix-closed; that is, if $vw \in D$, then $v \in D$. Thus, it always contains the empty path.

A *role tree* is a mapping $t : D \to \mathcal{P}(\mathcal{S})$ from a tree domain D into the powerset $\mathcal{P}(\mathcal{S})$ of $\mathcal{S}$. The paths in the domain of a role tree represent the nodes of the tree; the empty path represents its root. The letters s and t are used to denote role trees.

When convenient, we may consider a role tree t as a relation, i.e., $t \subseteq \mathcal{F}^* \times \mathcal{P}(\mathcal{S})$, and write $(w, S) \in t$ instead of $t(w) = S$. (Clearly, a relation $t \subseteq \mathcal{F}^* \times \mathcal{P}(\mathcal{S})$ is a role tree if and only if

$D = \{w \mid \exists S : (w, S) \in t\}$ is a tree domain and t is relational). As relations, i.e., as subsets of $\mathcal{F}^* \times \mathcal{P}(\mathcal{S})$, role trees are partially ordered by set inclusion. We say that s is *smaller than* (or, *is a prefix-subtree of*; or, *subsumes*; or, *approximates*) t if $s \subseteq t$.

The *subtree* wt of a role tree t at one of its nodes w is the role tree defined by (as a relation)

$$wt := \{(v, S) \mid (wv, S) \in t\}$$

If D is the domain of t, then the domain of wt is the set $w^{-1}D = \{v \mid wv \in D\}$. Thus, wt is given as the mapping $wt : w^{-1}D \to \mathcal{P}(\mathcal{S})$ defined on its domain by $wv(t) = t(wv)$. A role tree s is called a *subtree* of a role tree t if it is a subtree $s = wt$ at one of its nodes w, and a direct subtree if $w \in \mathcal{F}$.

A role tree t with domain D is called *rational* if (1) t has only finitely many subtrees and (2) t is finitely branching; that is: for every $w \in D, w\mathcal{F} \cap D = \{wf \in D \mid f \in \mathcal{F}\}$ is finite. Assuming (1), the condition (2) is equivalent to saying that there exist finitely many roles $f_1, \ldots, f_n$ such that $D \subseteq \{f_1, \ldots, f_n\}^*$.

A *path* p is a finite sequence of roles in $\mathcal{F}$. The *empty path* is denoted by ϵ and the free-monoid concatenation of paths p and p' as pp'; we have $\epsilon p = p\epsilon = p$. Given paths p and p', p' is called a *prefix of* p if $p = p'p''$ for some path p''. A tree domain is a non-empty prefixed-closed set of paths.

Definition 1 (Role Trees). *A role tree τ is a pair (D, L) consisting of a tree domain D and a partial labeling function $L : D \to \mathcal{S}$. Given a role tree τ, we write D_τ for its tree domain and L_τ for its labeling function. A role tree is called finite if its tree domain is finite, and infinite otherwise. We denote the set of all role trees by $\mathcal{R}$. If $p \in D_\tau$ we write as $\tau[p]$ the subtree of τ at path p which is formally defined by $D_{\tau[p]} = \{(p', S) \mid pp' \in D_\tau\}$ and $L_{\tau[p]} = \{(p', S) \mid (pp', S) \in L_\tau\}$.*

5.1.1 Syntax and Semantics of Ordering Constraints

In the following, we introduce the syntax and semantics of ordering constraints over role trees. We assume an infinite set (which we denote by $\mathcal{V}$) of tree variables ranged over by x, y, an infinite set (which we denote by $\tilde{V}$) of path variables ranger over by α, β, an infinite set $\mathcal{F}$ of *roles* ranged over by f, g, and an arbitrary multiset $\mathcal{S}$ of sorts denoted by S, T containing at least two distinct elements.

Syntax. An *ordering constraint* φ is defined by the following abstract syntax.

$$\varphi ::= x \leq y \mid S(\alpha, x) \mid x[v]y \mid x \sim y \mid \varphi_1 \wedge \varphi_2$$

where v is a role variable.

An ordering constraint is a conjunction of *atomic constraints* which are either *atomic ordering constraints* $x \leq y$, *generalized labeling constraints* $S(\alpha, x)$, *selection constraints* $x[v]y$, or *compatibility constraints* $x \sim y$.

For example, the complex constraint:

$$\mathsf{x[v]z} \wedge \mathsf{y[v]t} \wedge \mathsf{z} \sim \mathsf{t} \wedge \{\mathsf{DBMS}\}(\alpha, \mathsf{z}) \wedge \{\mathsf{DBMS}\}(\alpha, \mathsf{t})$$

expresses the fact that the pair x, y of trees have compatible subtrees via the same feature (here the valuation of the variable v), and such that both subtrees lead to the sort DBMS following the same path from the root of each.

Semantics. The signature of the structure contains the binary relation symbols $\leq$, $\sim$, and S(•,•) for every set of labels S, and for every role f a binary relation symbol $\bullet[f]\bullet$. The domain of the structure $\mathcal{R}$ is the set of possibly infinite role trees. The relation symbols are interpreted as follows:

$$
\begin{array}{lll}
\tau_1 \leq \tau_2 & \text{iff} & D_{\tau_1} \subseteq D_{\tau_2} \text{ and } L_{\tau_1} \subseteq L_{\tau_2} \\
\tau_1[v]\tau_2 & \text{iff} & D_{\tau_2} = \{p \mid fp \in D_{\tau_1}\} \text{ and } L_{\tau_2} = \{(p,S) \mid (fp,S) \in L_{\tau_1}\} \\
 & & \qquad \text{where } f = \sigma(v), \text{ with } \sigma \text{ a valuation} \\
S(\alpha,\tau) & \text{iff} & \mu(\alpha) \in D_\tau \text{ and } (\mu(\alpha),S) \in L_\tau \\
\tau_1 \sim \tau_2 & \text{iff} & L_{\tau_1} \cup L_{\tau_2} \text{ is a partial function (on } D_{\tau_1} \cup D_{\tau_2})
\end{array}
$$

where μ is a valuation from $\tilde{V}$ to the set of elements $\mathcal{F}^*$.

5.1.2 Satisfiability Test

We present a set of axioms valid for our constraint system and then interpret these axioms as an algorithm that solves the satisfiability problem of our constraint system.

Table 1 contains axioms schemes F1 - F6 that we regard as sets of axioms. The union of these sets of axioms is denoted by F. For instance, an axiom scheme $x \leq x$ represents the infinite set of axioms obtained by instantiation of the meta variable x. An axiom is either a constraint φ, an implication between constraints $\varphi \rightarrow \varphi'$, or an implication $\varphi \rightarrow false$.

F1.1	$x \leq x$
F1.2	$x \leq y \wedge y \leq z \rightarrow x \leq z$
F2	$x[v]x' \wedge x \leq y \wedge y[v]y' \rightarrow x' \leq y'$
F3.1	$x \sim x$
F3.2	$x \leq y \wedge y \sim z \rightarrow x \sim z$
F3.3	$x \sim y \rightarrow y \sim x$
F4	$x[v]x' \wedge x \sim y \wedge y[v]y' \rightarrow x' \sim y'$
F5	$S(\alpha,x) \wedge S'(\alpha,x) \rightarrow false$ for $S \neq S'$
F6	$S(\alpha,x) \wedge S'(\alpha,y) \wedge x \sim y \rightarrow false$ for $S \neq S'$

Table 1: Axioms of Satisfiability: F1-F6

The role tree structure $\mathcal{T}$ is defined as follows:

- The domain of $\mathcal{T}$ is the set of all role trees.
- $t \in A^{\mathcal{T}}$ if and only if $t(\epsilon) = A$ (t's root is labeled with the sort A).
- $(s,t) \in f^{\mathcal{T}}$ if and only if $f \in D_s$ and $t = fs$ (t is the subtree of s at f).

Proposition 1 *The structure $\mathcal{T}$ is a model of the axioms in* F.

Proof We prove the statement for the rules in F5 and F6.

F5) $S(\alpha, x) \wedge S'(\alpha, x)$ iff $\exists p \in D_x$ and $(p, S) \in L_x$ and $(p, S') \in L_x$. Or p is a path (a finite sequence of labels). This means that we cannot have simultaneously $(p, S) \in L_x$ and $(p, S') \in L_x$.

F6) $S(\alpha, x) \wedge S'(\alpha, y) \wedge x \sim y \leftrightarrow S(\alpha, x) \wedge S'(\alpha, y) \wedge \exists z(x \leq z \wedge y \leq z)$
$\rightarrow S(\alpha, x) \wedge S(\alpha, z) \wedge S'(\alpha, y) \wedge S'(\alpha, z)$
$\rightarrow false$ (according to F5, i.e., $S(\alpha, x) \wedge S'(\alpha, x) \rightarrow false$)

■

We define the size of a constraint φ to be the number of occurrences of roles,node labels, and variables in φ.

Proposition 2 *If φ is a constraint of size n then the algorithm (i.e., set of axioms F) which starts with φ as an input terminates in at most $2.n^2$ steps. Here, F1.1 and F3.3 apply to variables in φ only.*

Proof Note that the algorithm F does not add new variables to a set of constraints. The new constraints that can be added are of the form $x \sim y$, $x \leq y$, or $false$. The number of new constraints of the form $x \sim y$ that can be added is bounded by n^2. The number of new constraints of the form $x \leq y$ that can be added is also bounded by n^2. ■

Expressiveness. We show that our constraint system is strictly more expressive than $FT_{\leq}$ [46]. The feature constraints in the constraint system $FT_{\leq}$ are conjunctions of the following constraints, which are built from variables x, y, features f, and node labels a.

$$\varphi ::= x \leq y \mid a(x) \mid x[f]y \mid x \sim y \mid \varphi_1 \wedge \varphi_2$$

Proposition 3 *There is no constraint of $FT_{\leq}$ which cannot be expressed in our constraint system.*

Proof
Every constraint of $FT_{\leq}$ can trivially be expressed in our constraint system. The $FT_{\leq}$ constraint $a(x)$ can be expressed by $a(\epsilon, x)$ where ϵ is the empty path.

■

5.2 Path Constraints

The language of terms uses a countable set $\tilde{V}$ of variables called path variables, and denoted $\alpha, \beta, \ldots$.

Definition 2 *(**Path**) A path is a finite string of roles. We identify a label f (i.e., a role name in our case) with the string (f) consisting of a single role. We say that a path u is a* **prefix** *of a path v (written $u \dot{\prec} v$) if there is a non-empty path w such that $v = u.w$. Note that $\dot{\prec}$ is neither symmetric nor reflexive. We say that two paths u, v* **diverge** *(written $u \ddot{\amalg} v$) 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 $\ddot{\amalg}$ is a symmetric relation.*

Proposition 4 *Given two paths u and v, then exactly one of the relations $u \doteq v$, $u \dot{\prec} v$, $u \dot{\succ} v$ or $u \ddot{\amalg} v$ holds.*

Definition 3 *(**Path Term**) A path term, denoted p, q, ..., is either a path variable α or a concatenation of path variables $\alpha.\beta$.*

5.2.1 Syntax and Semantics of Path Constraints

Definition 4 (Path Constraint) *The set of atomic path constraints is given by:*

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

L is a regular expression denoting a regular language $\mathcal{L}(L) \subseteq \mathcal{F}^+$, where $\mathcal{F}$ is a set of roles.

For example, the conjunction

$$\alpha \dot{\amalg} \alpha' \wedge \beta \dot{\prec} \alpha \wedge \beta \dot{\prec} \alpha' \wedge \beta \in \mathsf{video.sequence}$$

Says that the two paths α and α' have the same prefix video.sequence, but they are divergent.

An **interpretation** $\mathcal{I}$ is a standard first order structure, where every role $f \in \mathcal{F}$ is interpreted as a binary relation $f^{\mathcal{I}}$. A **valuation** $\nu_{\tilde{V}}$ 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 $\nu_{\tilde{V}}$ is defined as follows:

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

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

6 Constraint-Based Query Language for Semistructured Data

We now present a construction that, given the constraint languages, let us call them $\mathcal{C}$ (for ordering constraints over role trees) and $\mathcal{C}'$ (for path constraints) and a set $\mathcal{R}$ of relation symbols, extends $\mathcal{C}$ and $\mathcal{C}'$ to a constraint query language $\mathcal{R}(\mathcal{C}, \mathcal{C}')$. Hence, we view our query language as consisting of two constraint languages on top of which relations can be defined by definite clauses.

6.1 Syntax

We define the predicate symbol **tree** to denote semistructured data represented as trees.

We reason about semistructured data by a program P which contains a set of rules defining ordinary predicates. The rules are of the form:

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

for some $n \geq 0$ and $m \geq 0$, where $\bar{X}, \bar{Y}_1, \ldots, \bar{Y}_n$ are tuples of tree variables or path variables. We require that the rules are safe, i.e., a variable that appears in $\bar{X}$ must also appear in $\bar{Y}_1 \cup \ldots \cup \bar{Y}_n \cup$ {path variables and tree variables occurring in $c_1, \ldots, c_m$}. The predicates $L_1, \ldots, L_n$ may be either *tree* or ordinary predicates. $c_1, \ldots, c_m$ are ordering constraints ($\mathcal{C}$-constraints) or path constraints

($\mathcal{C}'$-constraints). In the following, we use the term (positive) atom to make reference to predicates $L_1, \ldots, L_n$.

6.2 Semantics

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

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

1. A set $\mathcal{D}$ that is the union of two pair-wise disjoint sets:
   - $\mathcal{D}_1$: a set of tree domains
   - $\mathcal{D}_2$: a set of roles
2. A set $\mathcal{V}$ of variables called tree variables, and denoted $x, y, \ldots$
3. A set $\tilde{V}$ of variables called path variables, and denoted $\alpha, \beta, \ldots$.

We call $\mathcal{V} \cup \mathcal{D}$ first order terms. Let $\hat{V} = \mathcal{V} \cup \tilde{V}$.

6.2.1 Model-theoretic semantics

Let var_1 be a countable function that assigns to each syntactical expression a subset of $\mathcal{V}$ corresponding to the set of tree variables occurring in the expression, and var_2 be a countable function that assigns to each expression a subset of $\tilde{V}$ corresponding 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 $\mathcal{D}_2$ of roles, the extension of $\mathcal{D}_2$, written $\mathcal{D}_2^{ext}$, is the set of path expressions containing the following elements:*

- *each element in $\mathcal{D}_2$*
- *for each ordered pair p_1, p_2 of elements of $\mathcal{D}_2^{ext}$, 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 $\mathcal{D}_1 \cup \mathcal{D}_2$. 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 $\mathcal{D}_1 \cup \mathcal{D}_2^{ext}$.*

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 $\mathcal{D}_1$. A valuation ν_2 is a total function from $\tilde{V}$ to the set of elements $\mathcal{D}_2^{ext}$. 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 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 the 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.

6.2.2 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 two countable disjoint sets: a set of tree domains ($\mathcal{D}_1$), and a set of roles ($\mathcal{D}_2$). A path expression is an element of $\mathcal{D}_2^{ext}$. We define $\mathcal{D}^{ext} = \mathcal{D}_1 \cup \mathcal{D}_2^{ext}$.

At the iteration 1, only paths of length[3] 1 (i.e., simple roles, elements of $\mathcal{D}_2$) are considered during rules triggering. At iteration k, we consider paths of length less or equal to k. It is clear that the paths occurring in the final result are of length less or equal to the length of the longest path that can be found in all the tree domains.

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

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

[3]The length of a path is the number of roles composing the path.

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 $T_P(\mathcal{I}_1) \subseteq T_P(\mathcal{I}_2)$*

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 $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, ... 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 $\mathcal{D}^{ext}_{\mathcal{I}}$ since ν is based on $\mathcal{D}^{ext}_{\mathcal{I}}$. 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 $\mathcal{D}^{ext}_{\mathcal{I}_l}$ $\forall i \in [1,n]$ 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 $\mathcal{I}$. ■

6.3 Example

Let us give some simple examples of queries. Consider a company selling products. Figure 4 shows a fragment of data that the company has at its disposal. Different departments use different kinds of data according to their needs. For example, one department has data of type (a), another of type (b) and another of type (c). This data is stored in a warehouse intended to be queried and browsed.

The query "find all catalogs making reference to a product sold by a supplier located in Milano" can be expressed by the following rule:

$$answer(x) \leftarrow tree(x) \| \{Milano\}(\alpha.city, x)$$

In this example, α is a path variable, and x is a tree variable.

The query "find all pairs of catalogs making reference to a product of category *Camera* sold by suppliers in Italy and USA" can be expressed as:

$$answer(x, y) \leftarrow tree(x), tree(y) \| \{Italy\}(\alpha.country, x), \{USA\}(\alpha.country, y), \{Camera\}(\beta, x), \{Camera\}(\beta, y), \beta \dot{\in} product.category$$

If we want to compute all pairs of trees (y, x) such that y is a component (i.e., subtree) of x then we can use the following program:

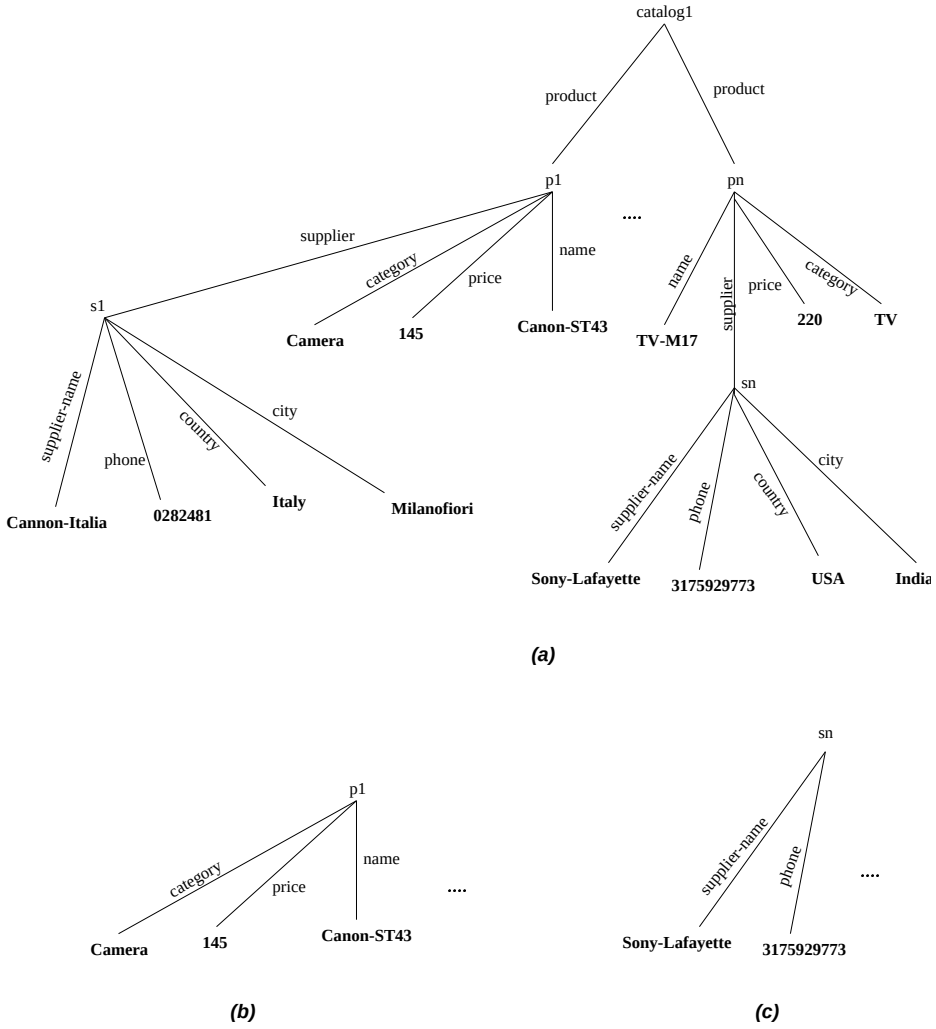

Figure 4: A fragment of our catalog, product and supplier database.

$$component(y, x) \leftarrow tree(x), tree(y) \| x[v]y$$
$$component(y, x) \leftarrow component(z, x), component(y, z)$$

7 Constraints Relaxation

When optimal solutions (i.e., those data trees matching the query tree) cannot be obtained, one may be interested in finding suboptimal solutions by relaxing some constraints.

According to the semantics of our query language, the satisfiability test for the constraint part of queries assumes ordered inclusion of trees, leading to exact matching of paths and sorts. For the need of approximate query answers, we have to deal with *unordered inclusion* of trees, and set constraints.

The tree, which can be a query tree, on the left of figure 5 is not perfectly embedded in the data tree on the right. The *c1* node in the data tree is skipped.

Definition 13 (Embedding) *A function f from the set Q of query nodes into the set D of data nodes is called an embedding if for all $q_i, q_j \in Q$:*

1. $f(q_i) = f(q_j) \Leftrightarrow q_i = q_j$

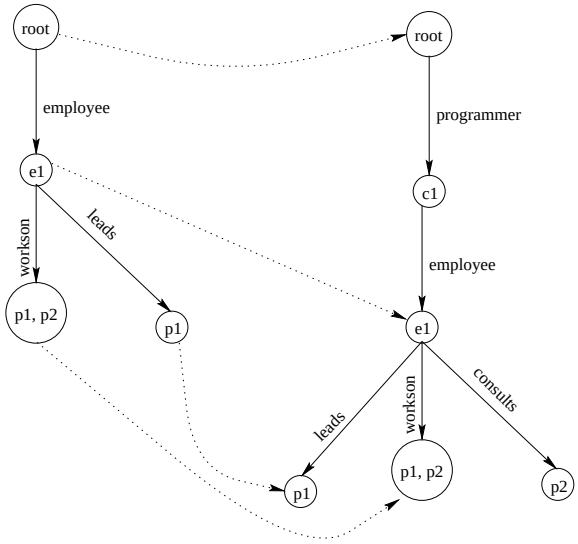

Figure 5: Unordered inclusion of trees.

2. $label(q_i) = label(f(q_i))$

3. *q_i is the parent of q_j with e the label of the edge from q_i to $q_j \Leftrightarrow f(q_i)$ is an ancestor of $f(q_j)$ with e the label of an edge from $f(q_i)$ to $f(q_j)$*

Let $p = a_1 \dots a_n$ be a path. We write $a_i \in p$ to denote the fact that a_i is a component of p. We write $a_i <_p a_j$ when $i < j$. That is, a_i is an ancestor of a_j in p.

Let $p = a_1 \dots a_n$ and $q = b_1 \dots b_m$ be two paths. We say that p is embedded in q, and we write $p \ll q$ iff

1. $n \leq m$

2. $\forall a_i \in p,\ a_i \in q\ (i \in [1, n])$

3. $\forall a_i, a_j \in p,\ a_i <_p a_j \Rightarrow a_i <_q a_j$

Let Q be a query of the form $L \| C$ where L is the body part and C is the constraint part. Assume that C contains a constraint of the form $S(p, x)$, where S is a set and p is a path. In the case Q doesn't return an exact answer, then rewrite the constraint C by replacing $S(p, x)$ by $X(\alpha, x), X \subseteq S, \alpha \ll p$, where X is a new set variable, and α is a new path variable.

$X \subseteq S$ (also called set constraint) and $\alpha \ll p$ are *weak* constraints. The evaluation of the new query, under the semantics of $\subseteq$ and $\ll$ will return a set of approximate answers.

To summarize, let $L \| C$ be a query, where C is its constraint part. Then, If C contains:

- the constraint $S(\bullet, x)$, where $\bullet$ stands for a path or a path variable, then replace $S(\bullet, x)$ by $X(\bullet, x), X \subseteq S$, with X a new set variable.
- the constraint $\bullet(p, x)$, where p is a path, then replace $\bullet(p, x)$ by $\bullet(\alpha, x), \alpha \ll p$, with α a new path variable.

8 Relation to XML

In this section, we show how the proposed framework can support querying XML data. We have to show how XML documents can be coded with role trees. We use a restricted structure of XML documents. However, the framework applies to the complete structure as defined in [49, 15, 28].

XML[4] is a textual representation of data. The basic component in XML is the *element*, that is, a piece of text bounded by matching tags such as < faculty > and < /faculty >. Inside an element we may have "raw" text, other elements, or a mixture of the two. Consider the following XML example:

```
< faculty >
    < name > Clint < /name >
    < room > 420 < /room >
    < email > crm@cs.edu < /email >
< /faculty >
```

An expression such as < faculty > is called a start-tag and < /faculty > an end-tag. Start- and end-tags are also called markups. Such tags must be balanced; that is, they should be closed in inverse order to that in which they are opened, like parentheses. Tags in XML are defined by users; there are no predefined tags, as in HTML. The text between a start-tag and the corresponding end-tag, including the embedded tags, is called an *element*, and the structures between the tags are referred to as the *content*. The term *subelement* is also used to describe the relation between an element and its component elements. Thus < email > ... < /email > is a subelement of < faculty > ... < /faculty > in the example above. As with semistructured data, we may use repeated elements with the same tag to represent collections. The following is an example in which several < faculty > tags occur next to each other.

```
< people >
   < faculty >
       < name > Clint < /name >
       < room > 420 < /room >
       < email > crm@cs.edu < /email >
   < /faculty >
   < faculty >
       < name > Marion < /name >
       < room > 319 < /room >
       < email > mj@cs.edu < /email >
   < /faculty >
< /people >
```

The basic XML syntax is perfectly suited for describing semistructured data. Recall the syntax for semistructured data expressions. The simple XML document

```
< faculty >
    < name > Clint < /name >
```

[4]For more details, see [2].

< room > 420 < /room >
< email > crm@cs.edu < /email >
< /faculty >

has the following representation as a semistructured data expression:

$$\{\text{faculty} : \{\text{name} : \text{"Clint"}, \text{room} : 420, \text{email} : \text{"crm@cs.edu"}\}\}$$

There is a subtle distinction between an XML element and a semistructured data expression. A semistructured data expression is a set of label/subtree pairs, while an element has just one top-level label. XML denotes graphs with labels on nodes[5]. To be able to apply the proposed framework, we consider that an XML document is represented by a role tree of a specific form. The only two edge labels are subelement and value. Figure 6 illustrates our representation for the XML data above.

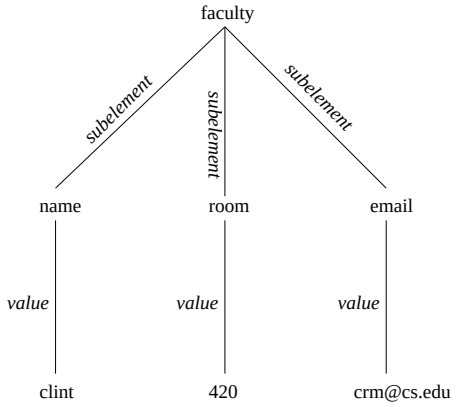

Figure 6: Our labeled-tree representation for XML data.

9 Related Work

We discuss the relationship of our work to *query languages for semistructured data* and *path queries with constraints*. Please note that the proposed frameworks and results by others are all different from ours.

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.

[5] While semistructured data expressions denote graphs with labels on edges.

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.

[43] 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 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.

[34] 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 query languages for the World-Wide Web that have emerged recently, e.g., W3QS [38], which focuses on extensibility, and *WebLog* [39] which is based on a Datalog-like syntax. Additional relevant work includes query languages for hypertext structures, e.g., [13, 26, 45, 42], and work on integrating SGML [30] 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 [47] and UniSQL/M [37] are designed to integrate data in object-oriented and relational databases. At the other end of the spectrum, systems such as GAIA [48] and ACL/KIF [29] provide uniform access to data with minimal structure.

[38] 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 [51] 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. The constraint $p = q$ is also allowed in our constraint language. Constraints such as $\alpha \dot{\prec} \beta$ and $\alpha \dot{\amalg} \beta$, and ordering constraints cannot be expressed as word constraints of [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 $\alpha \dot{\amalg} \beta$, or the ordering constraints.

Please note that no other work considered the ordering constraints we propose in this paper.

10 Conclusion

In this paper we have shown how two classes of constraints can be combined to make a flexible query language for semistructured data.

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 ordering and 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 ordering and 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 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 ordering and 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.

As for the future: (1) We want to develop an SQL-like language based on the formal model we have developed in order to provide a more user-friendly syntax; (2) In this paper, sorts appearing in ordering constraints (i.e., constraints of the form $\mathsf{S}(\alpha, \mathsf{x})$) are constants (i.e., S is a set of constants). In order our query language to be more flexible, it will be useful to consider sorts as first class values. As a

consequence, we will allow constraints of the form $\mathsf{Z}(\alpha, \mathsf{x})$, where Z is a variable ranging over sets of sorts; (3) Another direction of research consists in investigation the relationship between XPath (i.e., XML Path Language) [24] and our constraints.

References

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

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on The Web, From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Francisco, California, 2000.

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

[4] Serge Abiteboul and Victor Vianu. Queries and Computation on the Web. In Foto N. Afrati and Phokion 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] Serge Abiteboul and Victor 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] Hassan Aït-Kaci. An Algebraic Semantics Approach to the Effective Resolution of Type Equations. *Theoretical Computer Science*, 45:293–351, 1986.

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

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

[9] Hassan Aït-Kaci, Andreas Podelski, and Gert 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 *Proceedings of the First International Conference on Deductive and Object Oriented Databases (DOOD'89), Kyoto, Japan*, pages 40–57, 1989.

[11] Franz Baader, Hans J. Bückert, Bernhard Nebel, Werner Nutt, and Gert 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] Rolf 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(1/2):1–37, 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] Bret Bos. The XML Data Model. http://www.w3.org/XML/Datamodel.html, 1999.

[16] Peter Buneman. Semistructured Data. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'97), Tucson, Az, USA*, pages 117–121, 1997.

[17] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan 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] Peter Buneman, Wenfei Fan, and Scott 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] Hans-Jürgen Bürckert. A Resolution Principle for Constrained Logics. *Artificial Intelligence*, 66:235–271, 1994.

[20] R. G. G. Cattell. The Object Database Standard: ODMG-93. Morgan 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 (TKDE'96)*, 8(4):515–521, 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] James Clark and Steve DeRose. XML Path Language (XPath). Technical report, http://www.w3.org/TR/xpath, November 1999.

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

[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*, pages 269–292, November 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] Mary Fernandez and Jonathan Robie. XML Query Data Model. http://www.w3.org/TR/query-datamodel/, May 11 2000.

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

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

[31] Mohand-Saïd Hacid, Cyril Decleir, and Jacques Kouloumdjian. A Database Approach for Modeling and Querying Video Data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(5):729–750, 2000.

[32] Markus Höhfeld and Gert Smolka. Definite Relations over Constraint Languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988.

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

[34] Yaron Kanza, Werner Nutt, and Yehoshua 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.

[35] Ronald M. Kaplan and Joan 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.

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

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

[38] David Konopnicki and Oded Shmueli. W3QS: A Query System for the World-Wide Web. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Proceedings of the 21th International Conference on Very Large Databases (VLDB'95), Zurich, Switzerland*, pages 54–65. Morgan Kaufmann, September 1995.

[39] 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)*, pages 12–21, February 1996.

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

[41] 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.

[42] A. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal of Computing*, 24(6):1235–1258, 1995.

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

[44] Alberto O. Mendelzon and Tova 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.

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

[46] Martin Müller, Joachim Niehren, and Andreas Podelski. Ordering Constraints over Feature Trees. In Gert Smolka, editor, *Principles and Practice of Constraint Programming - CP97, Third International Conference (CP'97), Linz, Austria*, LNCS 1330, pages 297–311. Springer Verlag, 1997.

[47] 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.

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

[49] Jonathan Robie, Joe Lapp, and David Schach. XML Query Language (XQL). http://www.w3.org/TandS/QL/QL98/pp/xql.html, September 1998.

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

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

[52] 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.

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

[54] Gert Smolka and Ralf Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, April 1994.

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

[56] J. Widom 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.