Queries and Constraints on Semi-Structured Data

Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lastname@dis.uniroma1.it,
http://www.dis.uniroma1.it/~*lastname*

Abstract. We extend the model for semi-structured data proposed in [4], where both databases and schemas are represented as graphs, with the possibility of expressing different types of constraints on the nodes of the graphs, and defining queries which are used to select graphs from a database. We show that reasoning tasks at the basis of query optimization, such as schema subsumption, query-schema comparison, query containment, and query satisfiability, are decidable.

1 Introduction

The ability to represent data whose structure is less rigid and strict than in conventional databases is considered a crucial aspect in modern approaches to information systems, and is important in many application areas, such as biological databases, digital libraries, and data integration [13, 1, 4, 12, 8].

Recent proposals of models for semi-structured data represent data as graphs with labeled edges, where information on both the values and the schema of data are kept. In particular, we base our work on the BDFS data model [4], where graphs are used to represent both portions of a database (called ground graphs) and schemas. The former have edges labeled by data, and the latter have edges labeled by formulae of a decidable and complete first-order theory $\mathcal{T}$ over a fixed, finite universe $\mathcal{U}$.

A *ground graph* is a rooted connected graph whose edges are labeled with formulae of the form $(\mathit{self} = a)$[1]. A *graph schema* is a rooted connected graph whose edges are labeled with unary formulae of $\mathcal{T}$. Note that a *ground graph* is a special case of *graph schema.* A *semi-structured database* (or simply *database*) is a finite set of graphs (where each one is either a ground graph or a graph schema).

The notion of a ground graph g conforming to a schema S is given in terms of a special relation, called simulation, between the two graphs. Given a ground graph g and a schema S, a *simulation* from g to S is a binary relation $\trianglelefteq$ from the nodes of g to those of S such that $u \trianglelefteq u'$ implies that for each edge $u \xrightarrow{a} v$ in

[1] For each constant a of $\mathcal{T}$, $(\mathit{self} = a)(a')$ is **true** if and only if $a = a'$.

g, there exists an edge $u' \xrightarrow{p} v'$ in S such that $\mathcal{T} \models p(a)$, and $v \trianglelefteq v'$. A ground graph g *conforms to a* schema S, in notation $g \preceq S$, if there exists a *simulation* from g to S such that $root(g) \trianglelefteq root(S)$. Observe that the notion of simulation is less rigid than the usual notion of satisfaction, and suitably reflects the need of dealing with less strict structures of data.

For several tasks related to data management, it is important to be able to check subsumption between two schemas, i.e., to check whether every ground graph conforming to one schema always conforms to another schema. Given two schemas S and S', S' *subsumes* S, in notation $S \sqsubseteq S'$, if for every ground graph g, $g \preceq S$ implies $g \preceq S'$. S' and S are *equivalent* if both $S \sqsubseteq S'$ and $S' \sqsubseteq S$. In [4], an algorithm is presented for checking subsumption (and conformance, being a ground graph a special case of schema). The algorithm essentially looks for the greatest simulation between the nodes of the two schemas, and works in polynomial time with respect to the size of the two schemas.

In [4] the issue of extending the model with different types of constraints is raised. Indeed, in BDFS all the properties of the schema are expressed in terms of the structure of the graph, and the possibility of specifying additional constraints, such as existence of edges, is precluded. In this paper we extend the framework of [4] presenting the following contributions:

- We extend BDFS schemas with constraints (Section 2). The basic idea is to express constraints in terms of formulae associated to nodes of the schema. A formula on a node u imposes a condition that, for every ground graph g conforming to S, must be satisfied by every node of g simulating u. We consider different types of constraints, and we discuss how the expressive power of the constraint language influences the complexity of subsumption checking. In particular, we show that by adding edge-existence and functionality constraints the complexity of subsumption remains polynomial.
- We introduce a basic form of queries (Section 3), called *graph selection queries*, which are used to select graphs from a database The query language presented here represents a basic building block of a full-featured query language and has been designed on one hand to express sophisticated fixpoint properties of graphs, and on the other hand to keep several interesting reasoning tasks decidable. These reasoning tasks, such as comparing queries and schemas or checking containment between queries, are at the basis of query optimization techniques applicable to a more expressive query language.

2 Schemas with Constraints

We address the problem of extending the BDFS data model in order to express constraints on a schema. We conceive a constraint for a schema S as a formula associated to a node u of the schema. The formula is expressed in a certain language $\mathcal{L}$, and its role is to impose a condition that, for every ground graph g conforming to S, must be satisfied by every node of g simulating u.

A *schema with $\mathcal{L}$-constraints*, or simply *$\mathcal{L}$-schema*, is a schema where each node u is labeled by a formula $\mathcal{C}(u)$ of the constraint language $\mathcal{L}$. Given a ground graph g and an $\mathcal{L}$-schema S, a *simulation* from g to S is a binary relation $\trianglelefteq$ from the nodes of g to those of S such that $u \trianglelefteq u'$ implies that (1) u satisfies $\mathcal{C}(u')$, and (2) for each edge $u \xrightarrow{a} v$ in g, there exists an edge $u' \xrightarrow{p} v'$ in S such that $\mathcal{T} \models p(a)$, and $v \trianglelefteq v'$.

Apart from the new definition of simulation, the notions of conformance, subsumption and equivalence remain unchanged. We assume that $\mathcal{L}$ contains the formula $\top$, which is satisfied by every node of every ground graph. Therefore, we can view a ground graph g as an $\mathcal{L}$-schema where $\mathcal{C}(u) = \top$ for every node u of g. Thus, conformance is again a special case of subsumption.

Since constraints may contradict each other, or may even be incompatible with the structure of the graph, the notion of consistency becomes relevant (notice that a ground graph is always consistent). Given an $\mathcal{L}$-schema S, a node $u \in \mathit{Nodes}(S)$ is *consistent* if there is at least one ground graph which conforms to S', where S' is equal to S except that $\mathit{root}(S') = u$. S is *consistent*, if $\mathit{root}(S)$ is consistent. Moreover, two $\mathcal{L}$-schemas S_1 and S_2 are said to be *disjoint*, if there is no ground graph that conforms to S_1 and S_2.

In general, adding constraints to a schema leads to intractability of reasoning [7]. Here we consider a language $\mathcal{L}_l$ that allows for expressing interesting forms of constraints, and for which reasoning remains tractable. The main point is that we allow for expressing only local constraints, i.e., constraints on the edges directly emanating from a node. Formulae in $\mathcal{L}_l$ have the following syntax (γ, γ_1 and γ_2 denote constraints, and p denotes a formula of $\mathcal{T}$):

$$\gamma \;::=\; \top \;\mid\; \exists\mathsf{edge}\,(p) \;\mid\; \neg\exists\mathsf{edge}\,(p) \;\mid\; \exists^{\leq 1}\mathsf{edge}\,(p) \;\mid\; \gamma_1 \wedge \gamma_2$$

Intuitively, a constraint of the form $\exists\mathsf{edge}\,(p)$ on a node u, called *edge-existence constraint*, imposes that u has at least one outgoing edge $u \xrightarrow{a} v$ such that $\mathcal{T} \models p(a)$, while a constraint of the form $\exists^{\leq 1}\mathsf{edge}\,(p)$, called *functionality-constraint*, imposes that u has at most one such outgoing edge.

Our main result is that reasoning with schemas with local constraints can be done in polynomial time. In particular:

1. We have devised an algorithm for checking whether an $\mathcal{L}_l$-schema S is consistent. The algorithm is based on a function that first removes the non-existence constraints, and then removes all inconsistent nodes from S. The function runs in time polynomial in the size of S.
2. We have extended the algorithm in [4] in order to deal with local constraints. The basic idea of the algorithm is to look for a simulation between the two schemas by constructing a relation R as the Cartesian product of the two sets of nodes, and then removing from R all the pairs (u, u') for which no relation $\trianglelefteq$ satisfying condition (2) above may exist. The algorithm runs in time polynomial in the size of the two $\mathcal{L}_l$-schemas.
3. We have shown that our technique can also be used to perform in polynomial time the tow following tasks: computing the *Least Upper Bound* (LUB) [4] of two $\mathcal{L}_l$-schemas, and checking whether two $\mathcal{L}_l$-schemas are disjoint.

3 Graph Selection Queries

In general, query languages on semi-structured data are constituted by two components: one for selecting graphs, and another one for restructuring the selected graph to produce the actual answer [5, 3, 9, 2]. Here we introduce a basic form of queries, which we call *graph selection queries (gs-queries)*, which deal only with the selection part. The language of gs-queries allows for expressing sophisticated fixpoint properties of graphs. Furthermore it has been carefully designed in order to keep several interesting reasoning tasks decidable, such as checking query satisfiability, checking containment or disjointness between queries, and comparing queries and schemas.

Observe that the unit retrieved by a gs-query is a graph, whereas there is no means to further manipulate specific data from the retrieved graph (see for example [10]). Therefore our language cannot be considered a full-featured query language, such as UnQL [5], but should rather be regarded as providing basic building blocks for querying semi-structured data, to be exploited in query processing for improving evaluation performance.

In the rest of the paper, we deal only with $\mathcal{L}_l$-schemas, which we simply call schemas. The language for expressing graph selection queries has the following syntax (p denotes a formula of $\mathcal{T}$, n a positive integer, and X a node variable)

$$\begin{array}{l} \textit{node formulae}\colon N ::= X \mid \exists^{\geq n}\mathsf{edge}\,(E) \mid \neg N \mid N_1 \wedge N_2 \mid \mu X.N \\ \textit{edge formulae}\colon E ::= p \mid \mathsf{to}(N) \mid \neg E \mid E_1 \wedge E_2 \end{array}$$

with the restriction that every free occurrence of X in $\mu X.N$ is in the scope of an even number of negations[2].

Let g be a ground graph. A *valuation* ρ on g is a mapping from node variables to subsets of $\mathit{Nodes}(g)$. We denote by $\rho[X/\mathcal{N}]$ the valuation identical to ρ except for $\rho[X/\mathcal{N}](X) = \mathcal{N}$. For each node $u \in \mathit{Nodes}(g)$, we define when *u satisfies a node formula N under a valuation ρ*, in notation $\rho, u \models N$, as follows:

$$\begin{array}{lll} \rho, u \models X & \text{iff} & u \in \rho(X) \\ \rho, u \models \exists^{\geq n}\mathsf{edge}\,(E) & \text{iff} & \#\{u \xrightarrow{a} v \in \mathit{Edges}(g) \mid \rho, u \xrightarrow{a} v \models E\} \geq n \\ \rho, u \models \neg N & \text{iff} & \rho, u \not\models N \\ \rho, u \models N_1 \wedge N_2 & \text{iff} & (\rho, u \models N_1) \;\wedge\; (\rho, u \models N_2) \\ \rho, u \models \mu X.N & \text{iff} & \forall \mathcal{N} \subseteq \mathit{Nodes}(g). \\ & & \quad (\forall v \in \mathit{Nodes}(g).\rho[X/\mathcal{N}], v \models N \supset \rho[X/\mathcal{N}], v \models X) \\ & & \qquad \supset \rho[X/\mathcal{N}], u \models X \end{array}$$

where

$$\begin{array}{lll} \rho, u \xrightarrow{a} v \models p & \text{iff} & \mathcal{T} \models p(a) \\ \rho, u \xrightarrow{a} v \models \mathsf{to}(N) & \text{iff} & \rho, v \models N \\ \rho, u \xrightarrow{a} v \models \neg E & \text{iff} & \rho, u \xrightarrow{a} v \not\models E \\ \rho, u \xrightarrow{a} v \models E_1 \wedge E_2 & \text{iff} & (\rho, u \xrightarrow{a} v \models E_1) \wedge (\rho, u \xrightarrow{a} v \models E_2) \end{array}$$

[2] This is the usual *syntactic monotonicity* constraint typical of fixpoint logics, that guarantees the monotonicity of the fixpoint operator.

Given a graph G (either a ground graph or a schema) and a closed node formula N, we say that G *satisfies* N, in notation $G \sqsubseteq N$, if for every ground graph g conforming to G, $root(g) \models N$. It is easy to see that if g is a ground graph and N is a node formula, then $g \sqsubseteq N$ if and only if $root(g) \models N$.

A *graph selection query (gs-query)* Q is a closed node formula. The evaluation of Q over a database DB returns the set $Q(DB)$ of all consistent graphs $G \in DB$ such that $G \sqsubseteq Q$. A gs-query Q is *satisfiable* if there exists a database DB such that $Q(DB)$ is non-empty. Given two gs-queries Q_1 and Q_2, Q_1 is *contained* in Q_2 if for every database DB, $Q_1(DB) \subseteq Q_2(DB)$, and Q_1 is *disjoint* from Q_2 if for every database DB, $Q_1(DB) \cap Q_2(DB) = \emptyset$.

On the basis of a polynomial reduction of satisfiability of a gs-query to satisfiability in a variant of modal mu-calculus [11], we were able to prove the following: Checking a gs-query for satisfiability and checking containment and disjointness between two gs-queries are EXPTIME-complete problems [6].

References

[1] S. Abiteboul. Querying semi-structured data. In *Proc. of ICDT-97*, pages 1–18, 1997.

[2] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, and J. S. Guido Moerkotte. Querying documents in object databases. *Int. J. on Digital Libraries*, 1(1):5–19, 1997.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[4] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT-97*, pages 336–350, 1997.

[5] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. of ACM SIGMOD*, pages 505–516, 1996.

[6] D. Calvanese, G. De Giacomo, and M. Lenzerini. Queries and constraints on semi-structured data. Technical Report 13-98, Dip. di Inf. e Sist., Univ. di Roma "La Sapienza", 1998.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. What can knowledge representation do for semi-structured data? In *Proc. of AAAI-98*, pages 205–210, 1998.

[8] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of ACM SIGMOD*, pages 414–425, 1998.

[9] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[10] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proc. of VLDB-97*, pages 436–445, 1997.

[11] D. Kozen. Results on the propositional μ-calculus. *Theor. Comp. Sci.*, 27:333–354, 1983.

[12] A. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[13] D. Quass, A. Rajaraman, I. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. of DOOD-95*, pages 319–344. Springer-Verlag, 1995.