Local Constraints in Semistructured Data Schemas

Andrea Calì, Diego Calvanese, Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{cali,calvanese,lenzerini}@dis.uniroma1.it

Abstract. Recently, there have been several proposals of formalisms for modeling semistructured data, which is data that is neither raw, nor strictly typed as in conventional database systems. Semistructured data models are graph-based models, where graphs are used to represent both databases and schemas. We study the basic problem of schema subsumption, which amounts to check whether all databases conforming to a schema also conform to another schema, in the presence of constraints, which are used to enforce additional conditions on databases. In particular, we study the relationship between various constraint languages and the basic property of locality, which allows one to check subsumption between schemas in polynomial time in the number of nodes of the schemas. We show that locality holds when both numeric constraints and disjunction are added to a simple constraint language. On the other hand, locality is lost when we consider constraints both on outgoing and incoming edges of databases.

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 data modeling, and is important in many application areas, such as web information systems, biological databases, digital libraries, and data integration [21, 1, 5, 20, 16, 17].

Semistructured data is data that is neither raw, nor strictly typed as in conventional database systems [1]. Recently, several formalisms for modeling semistructured data have been proposed, such as OEM (Object Exchange Model) [2], and BDFS (Basic Data model For Semistructured data) [5]. In such formalisms, data is represented as graphs with labeled edges, where information on both the values and the schema of data are kept.

In particular, BDFS is an elegant graph-based data model, where graphs are used to represent both databases and schemas, the former with edges labeled by data, and the latter with edges labeled by formulae of a suitable logical theory. The notion of a database g conforming to a schema S is given in terms of a special relation, called *simulation*, between the two graphs. Roughly speaking, a simulation is a correspondence between the edges of g and those of S such that, whenever there is an edge labeled a in g, there is a corresponding edge in S labeled with a formula satisfied by a. 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 database conforming to one schema always

conforms to another schema. In [5] an algorithm for checking subsumption in BDFS is presented and its complexity is analyzed.

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. The problem of extending BDFS with different types of constraints has first been studied in [7, 8]. The basic idea is to express constraints in terms of formulae associated to nodes of the schema. In [7, 8] several languages for expressing constraints are presented and the complexity of the basic inference tasks (namely checking consistency, conformance, and subsumption) for such languages is studied. In particular, so called *locality* of constraints is identified as a crucial property that allows one to perform the basic inference tasks by means of several *local checks*, and thus retains efficiency with respect to the total size of the schemas. A simple constraint language with the property of locality, called $\mathcal{L}_{sl}$, is presented, which allows one to express existence and uniqueness of outgoing edges with certain properties.

In this paper we investigate the relationship between the expressive power of constraint languages and the ability to retain locality, thus keeping the complexity of the basic inference tasks over schemas tractable. In particular, we study the following extensions of the constraint language $\mathcal{L}_{sl}$:

- We add to $\mathcal{L}_{sl}$ more complex forms of numeric constraints, than simple existence and uniqueness constraints, in the spirit of cardinality constraints typically used in conceptual data models [3, 18, 22, 4] and knowledge representation formalisms [15]. We show that this extension preserves locality, and hence that subsumption can be verified in time polynomial in the total number of nodes of the two schemas. However, differently from the case of $\mathcal{L}_{sl}$, a single local check may require exponential time in the size of a constraint.
- In addition to numeric constraints, we add the possibility to express disjunctions of constraints, thus obtaining a full propositional constraint language. Such extension has the same computational complexity as for the case without disjunction. In particular, we show that locality holds, although we have to resort to more involved techniques.
- We extend $\mathcal{L}_{sl}$ with the ability to express constraints not only on outgoing edges but also on incoming edges, in line with expressive representation formalisms both for traditional databases [11, 10] and for semistructured data [12, 19]. Although at a first glance such extension does not seem to influence locality, we show that locality is actually lost. We show also that in this case the finite model property does not hold, i.e., a schema may admit only infinite databases.

2 Framework

We present now the formal model for semistructured data introduced in [5] and its extension with constraints studied in [7, 8], which are at the basis of our investigation.

2.1 Basic Data model For Semistructured data (BDFS)

The data model introduced in [5], which we call BDFS (*Basic Data model For Semistructured data*), is an edge-labeled graph model for semistructured data where edge-labels are formulae of a

first-order language $\mathcal{L}_\mathcal{T}$. The language $\mathcal{L}_\mathcal{T}$ is built over a set of predicates, including the equality predicate "=", and contains one constant for each element of a fixed (not necessarily finite) universe $\mathcal{U}$. Schemas and databases in BDFS always refer to a *complete* and *decidable* theory $\mathcal{T}$ on $\mathcal{U}$. In other words, for each closed formula $F \in \mathcal{L}_\mathcal{T}$, either $\mathcal{T} \models F$ or $\mathcal{T} \models \neg F$, and it is decidable to check whether $\mathcal{T} \models F$. We can consider $\mathcal{T}$ as (a possibly compact representation of) the set of first order formulae which are true (or equivalently, valid) for the elements of $\mathcal{U}$.

In BDFS, databases and schemas are modeled with rooted edge-labeled graphs. A *rooted edge-labeled graph* is a pair (H, r), where H is an edge-labeled graph and r is a distinguished node of H, called *root*, such that each node of H is connected to r through a directed path. Given a rooted edge-labeled graph $G = (H, r)$, we denote the root r with $\text{root}(G)$, the set of nodes of H with $\text{Nodes}(G)$, and the set of edges of H with $\text{Edges}(G)$. We denote an edge from node u_f to node u_s labeled by a with $u_f \xrightarrow{a} u_s$.

Definition 1. *Given a theory $\mathcal{T}$, a* schema for $\mathcal{T}$ *is a rooted edge-labeled graph whose edges are labeled with unary formulae of $\mathcal{T}$. A* database for $\mathcal{T}$ *is a rooted edge-labeled graph whose edges are labeled with constants of $\mathcal{T}$.*

In the following, for the sake of conciseness, we omit the explicit reference to the theory $\mathcal{T}$ and use simply *schema* (resp., *database*) instead of schema for $\mathcal{T}$ (resp., database for $\mathcal{T}$).

To establish if a database is coherent with a schema, or if a schema is more specific than another schema, we define the notions of conformance and subsumption, which in turn are based on the fundamental notion of simulation.

Definition 2. *A* simulation *from a database g to a schema S is a binary relation $\trianglelefteq$ from* $\text{Nodes}(g)$ *to* $\text{Nodes}(S)$ *such that, for each pair of nodes v in g and u in S with $v \trianglelefteq u$ the following holds: For each edge $v \xrightarrow{c} v_s$ in g there exists an edge $u \xrightarrow{\pi} u_s$ in S such that $\mathcal{T} \models \pi(c)$ and $v_s \trianglelefteq u_s$.*

A database g conforms *to a schema S, in notation $g \preceq S$, if there exists a simulation $\trianglelefteq$ from g to S such that* $\text{root}(g) \trianglelefteq \text{root}(S)$.

Definition 3. *A schema S_1* is subsumed by *a schema S_2, in notation $S_1 \sqsubseteq S_2$, if for every database g we have that $g \preceq S_1$ implies $g \preceq S_2$. Two schemas S_1 and S_2 are* equivalent*, in notation $S_1 \equiv S_2$, if both $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_1$.*

Notice that we can view each database as a schema, by considering a constant a labeling an edge as an abbreviation for the unary formula $\lambda x.x = a$. It is easy to see that a database g conforms to a schema S, iff g, viewed as a schema, is subsumed by S [5]. Therefore, conformance is a special case of subsumption and henceforth we concentrate on subsumption only.

In [5], an algorithm is presented for checking subsumption between two schemas. The algorithm essentially looks for the greatest simulation between the nodes of the two schemas, and works in time $O(m^{O(1)} \cdot t_\mathcal{T}(m))$, where m is the size of the two schemas and $t_\mathcal{T}(x)$ is the time needed to check whether a formula of size x is valid in $\mathcal{T}$. As argued in [5], it is meaningful to consider $\mathcal{T}$ not part of the input of the problem. Therefore, whenever $t_\mathcal{T}(m)$ may be assumed to be independent of m, $t_\mathcal{T}(m)$ can be replaced by a constant (e.g., when the size m of the two schemas is considerably smaller than the size of $\mathcal{T}$).

Example 1. In Figure 1(a) we present a schema modeling a university divided in faculties, which in turn are divided into departments; each faculty can have students, each department can have professors, and each student can have a professor as tutor and a set of exams she has given.

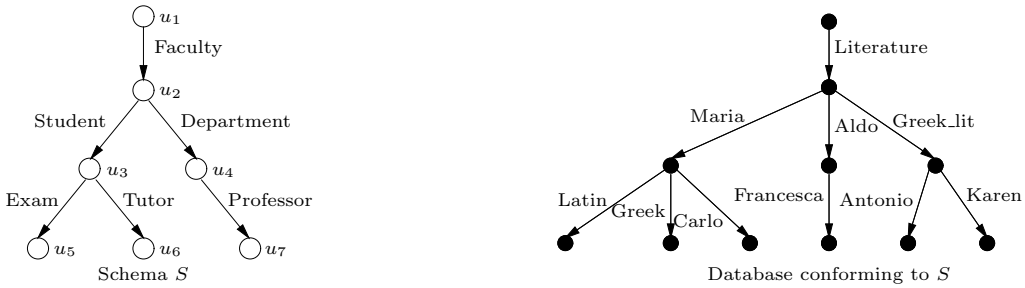


Fig. 1. A schema modeling a university and a database conforming to it

2.2 Schemas with Constraints

To overcome several limitations of the basic BDFS model, e.g., the ability to enforce the existence of edges, in [7, 8] it proposed to add to schemas the ability to express *constraints*. Constraints, which are attached to the nodes of a schema, are expressed in a certain *constraint language*, and impose additional conditions on the outgoing or incoming edges of the nodes of a database, wrt those already imposed by the schema. Given a constraint language $\mathcal{L}$, one can define when a node u of a database *satisfies* a constraint γ of $\mathcal{L}$, in notation $u \rightarrowtail \gamma$. Then the notion of simulation is extended to take into account also constraints, by requiring that a node u in a database satisfies the constraint associated to a node in the schema that u simulates.

Formally, given a constraint language $\mathcal{L}$, a schema (for $\mathcal{T}$) with $\mathcal{L}$-constraints, simply called *$\mathcal{L}$-schema*, is an edge and node labeled graph, such that the graph ignoring node labels is a schema, and node labels are constraints in $\mathcal{L}$. We denote with $\mathcal{C}(u)$ the constraint in $\mathcal{L}$ labeling node u.

Definition 4. *A* simulation *from a database g to an $\mathcal{L}$-schema S is a binary relation $\trianglelefteq$ from* Nodes(g) *to* Nodes(S) *such that, for each pair of nodes v in g and u in S with $v \trianglelefteq u$:*

1. *For each edge $v \xrightarrow{c} v_s$ in g, there exists an edge $u \xrightarrow{\pi} u_s$ in S such that $\mathcal{T} \models \pi(c)$ and $v_s \trianglelefteq u_s$.*
2. *v satisfies the constraint $\mathcal{C}(u)$.*

Conformance, subsumption, and equivalence can be defined as for schemas without constraints, considering the new definition of simulation. However, while in the absence of constraints every schema has at least one database that conforms to it[1], when we add constraints we also need to consider the notion of consistency.

Definition 5. *An $\mathcal{L}$-schema is said to be* consistent *if there exists a database that conforms to it. A node u of an $\mathcal{L}$-schema S is said to be* consistent *if there exists a database that conforms to S_u, where S_u is the $\mathcal{L}$-schema identical to S except for the root, which is u.*

As an example, consider the schema S, where S consists of a single node r and contains no edges. The constraint is $\exists\mathsf{edge}(\mathit{true})$ and expresses the existence of at least one outgoing edge. Such a schema is inconsistent, since for a database node simulating r, the constraint $\mathcal{C}(r)$ imposes the existence of an outgoing edge, but condition (1) of the simulation forbids the existence of such an edge.

[1] The empty database, which is the database consisting of a single node, conforms to every schema.

Syntax	Semantics
$v \rightarrowtail \top$	Always
$v \rightarrowtail \exists \mathsf{edge}(p)$	$\exists v \xrightarrow{c} v_s \in \mathrm{Edges}(g) \mid \mathcal{T} \models p(c)$
$v \rightarrowtail \nexists \mathsf{edge}(p)$	$\nexists v \xrightarrow{c} v_s \in \mathrm{Edges}(g) \mid \mathcal{T} \models p(c)$
$v \rightarrowtail \exists^{\leq 1} \mathsf{edge}(p)$	$\#\{v \xrightarrow{c} v_s \in \mathrm{Edges}(g) \mid \mathcal{T} \models p(c)\} \leq 1$
$v \rightarrowtail \gamma_1 \wedge \gamma_2$	$(v \rightarrowtail \gamma_1) \wedge (v \rightarrowtail \gamma_2)$

Table 1. Syntax and semantics of $\mathcal{L}_{sl}$ ($\#X$ denotes the cardinality of set X)

The ability to check for consistency of nodes turns out to be important also for checking subsumption, since inconsistent nodes need to be removed beforehand. On the other hand, since all meaningful constraint languages allow one to construct an inconsistent schema, schema inconsistency can be reduced to schema subsumption. Indeed, an $\mathcal{L}$-schema S is inconsistent if and only if it is subsumed by S_{inc}, where S_{inc} is an inconsistent schema.

2.3 Locality of Constraints

In [7, 8] several languages for expressing constraints are presented and the complexity of the basic inference tasks (namely checking consistency, conformance, and subsumption) for such languages is studied. In particular, so called *locality* of constraints is identified as a crucial property that allows one to perform the basic inference tasks by means of several *local checks*, and thus retain efficiency with respect to the total size of the schemas. A constraint $\mathcal{C}(u)$ labeling a node u of a schema is said to be *local* when it forces conditions that are local to the nodes simulating u. In other words, one can check whether u is consistent by considering only u with $\mathcal{C}(u)$ and the outgoing (and incoming[2]) edges of u (taking as nodes attached to such edges fresh nodes with no constraint on them), while ignoring all other nodes and edges in the database. A constraint language in which all constraints are local is called *local*.

We present now the language $\mathcal{L}_{sl}$, introduced in [7], for expressing simple types of local constraints. Such constraint language is at the basis of our investigation, and in Sections 3 and 4 we will consider extensions of $\mathcal{L}_{sl}$. Using constraints in $\mathcal{L}_{sl}$ one can impose the existence or non-existence of an outgoing edge satisfying a certain formula, the fact that there is at most one outgoing edge satisfying a certain formula, and conjunctions of such constraints. We call the constraints of $\mathcal{L}_{sl}$ *simple local constraints*. Formally, a constraint in $\mathcal{L}_{sl}$ is constructed according to the following abstract syntax:

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

where p is a unary formula of $\mathcal{T}$. The semantics of $\mathcal{L}_{sl}$ is given in Table 1, which specifies when a node v of a database g satisfies a constraint γ, in notation $v \rightarrowtail \gamma$.

Example 2. Considering again the schema of Figure 1, we would like to additionally enforce the following properties: (1) each faculty has at least one student and one department; (2) each student has at most one tutor; (3) each department has at least one professor. This can be done by using

[2] We will consider in Section 4 also constraints on incoming edges.

the following simple local constraints:

$\mathcal{C}(u_1) = \mathcal{C}(u_5) = \mathcal{C}(u_6) = \top$ $\quad$ $\mathcal{C}(u_3) = \exists^{\leq 1}\mathsf{edge}(\text{Tutor})$

$\mathcal{C}(u_2) = \exists\mathsf{edge}(\text{Student}) \wedge \exists\mathsf{edge}(\text{Department})$ $\quad$ $\mathcal{C}(u_4) = \exists\mathsf{edge}(\text{Professor})$

We call constraints of the form $\nexists\mathsf{edge}(p)$, which express nonexistence of an outgoing edge satisfying p, *negative constraints*. We observe that negative constraints can be eliminated from an $\mathcal{L}_{sl}$-schema without altering the semantics of the schema. Indeed, for a node u of an $\mathcal{L}_{sl}$-schema S for which $\mathcal{C}(u)$ contains the conjunct $\nexists\mathsf{edge}(p)$, we can remove $\nexists\mathsf{edge}(p)$ from $\mathcal{C}(u)$, provided that we conjoin all formulae labeling the outgoing edges of u with $\neg p$.

As shown in [7], to check whether a node u of an $\mathcal{L}_{sl}$-schema is consistent, it is sufficient to consider u with its constraint $\mathcal{C}(u)$ and the outgoing edges of u (while neglecting the constraints over the successors of u). In particular, it is possible to construct from $\mathcal{C}(u)$ and the formulae $r_1, \ldots, r_\ell$ labeling the outgoing edges of u a new formula π_u of polynomial size in $\mathcal{C}(u)$ and $r_1, \ldots, r_\ell$, such that $\mathcal{T} \models \pi_u$ if and only if u is consistent. Hence $\mathcal{L}_{sl}$ is local, and in [7] it is indeed shown that consistency and subsumption for $\mathcal{L}_{sl}$-schemas can be checked in polynomial time in the size of the schemas. More precisely, the algorithm for verifying subsumption between two schemas S_1 and S_2 first removes all inconsistent nodes from S_1 and S_2, and then tries to construct the greatest simulation between the nodes of S_1 and those of S_2. Both the removal of inconsistent nodes and the construction of the greatest simulation can be done by performing a number of validity checks that is polynomial in the size of the two schemas, and for which the result of each check depends only on a node and the adjacent edges. As shown in the next section for two meaningful extensions of $\mathcal{L}_{sl}$, this property holds in general when the constraint language is local, i.e., one can use a polynomial number of local validity checks to verify consistency and subsumption. The size of the formula to check for validity will typically be small wrt the total size of the two schemas. Hence, the total cost of subsumption can be kept low (e.g., polynomial) wrt the size of the two schemas, even when the cost of performing a single validity check is high wrt the size of the involved constraint and edge labels (e.g., the formula to check for validity is exponential in the constraint). For example, under the reasonable assumption that both the size of each constraint attached to a node and the number of outgoing edges for a node are logarithmic in the total size of a schema, checking subsumption can be done in polynomial time in the size of the two schemas.

On the other hand, the lack of locality for a constraint language $\mathcal{L}$ is a strong indication for intractability of the basic inference tasks on $\mathcal{L}$-schemas and databases. For example, in [7] the constraint language $\mathcal{L}_{\mathcal{ALE}}$ is presented, for which it is precisely the lack of locality that makes checking the consistency of an $\mathcal{L}_{\mathcal{ALE}}$-schema coNP-hard in the size of the schemas.

3 Local Constraints on Outgoing Edges

We now study two extensions of the constraint languages $\mathcal{L}_{sl}$ which we will prove to be local.

3.1 Extended Local Constraints

We now introduce a new constraint language $\mathcal{L}_{ext}$ of so called *extended local constraints*, inspired by cardinality constraints typically present in database models and Knowledge representation formalism [3, 18, 22, 4, 15]. The language $\mathcal{L}_{ext}$ is defined by the abstract syntax

$$\gamma \ ::= \ \top \ \mid \ \exists^{\leq n}\mathsf{edge}(p) \ \mid \ \exists^{\geq n}\mathsf{edge}(p) \ \mid \ \gamma_1 \wedge \gamma_2$$

Syntax	Semantics
$v \rightarrowtail \top$	Always
$v \rightarrowtail \exists^{\leq n}\mathsf{edge}(p)$	$\#\{v \xrightarrow{c} v_s \in \mathrm{Edges}(g) \mid \mathcal{T} \models p(c)\} \leq n$
$v \rightarrowtail \exists^{\geq n}\mathsf{edge}(p)$	$\#\{v \xrightarrow{c} v_s \in \mathrm{Edges}(g) \mid \mathcal{T} \models p(c)\} \geq n$
$v \rightarrowtail \gamma_1 \wedge \gamma_2$	$(v \rightarrowtail \gamma_1) \wedge (v \rightarrowtail \gamma_2)$

Table 2. Syntax and semantics of $\mathcal{L}_{ext}$

where p is a unary formula of $\mathcal{T}$, and n, called a *numeric index*, is a number in $\mathbb{N}$ (we assume $0 \in \mathbb{N}$) represented in *unary*.

The *semantics* of $\mathcal{L}_{ext}$ is shown in Table 2. Notice that $\mathcal{L}_{ext}$ extends $\mathcal{L}_{sl}$, since $\exists^{\leq 0}\mathsf{edge}(p)$ is equivalent to $\nexists\mathsf{edge}(p)$, and $\exists^{\geq 1}\mathsf{edge}(p)$ is equivalent to $\exists\mathsf{edge}(p)$. We point out that $\mathcal{L}_{ext}$ negative constraints,which are those of the form $\exists^{\leq 0}\mathsf{edge}(p)$, can be removed in the same way as for $\mathcal{L}_{sl}$.

Example 3. Let us consider again the schema of Figure 1. We would like to enforce, in addition to the properties specified in Example 2, also the property (4) a student cannot give more than 25 exams. This can be done by using constraints of $\mathcal{L}_{ext}$ as follows:

$\mathcal{C}(u_1) = \mathcal{C}(u_5) = \mathcal{C}(u_6) = \top$ $\qquad$ $\mathcal{C}(u_3) = \exists^{\leq 1}\mathsf{edge}(\mathrm{Tutor}) \wedge \exists^{\leq 25}\mathsf{edge}(\mathrm{Exam})$

$\mathcal{C}(u_2) = \exists^{\geq 1}\mathsf{edge}(\mathrm{Student}) \wedge \exists^{\geq 1}\mathsf{edge}(\mathrm{Department})$ $\qquad$ $\mathcal{C}(u_4) = \exists^{\geq 1}\mathsf{edge}(\mathrm{Professor})$

We now show that $\mathcal{L}_{ext}$ is local and therefore we can check the consistency of an $\mathcal{L}_{ext}$-schema by means of local checks for consistency of the nodes of the schema. Indeed, we present the function `cons`, which checks a node u for consistency by means of local checks. Let u have ℓ outgoing edges, labeled $r_1, \ldots, r_\ell$, and let the constraint $\mathcal{C}(u)$ over u be

$$\mathcal{C}(u) = \exists^{\leq n_1}\mathsf{edge}(p_1) \wedge \cdots \wedge \exists^{\leq n_s}\mathsf{edge}(p_s) \wedge \exists^{\geq m_1}\mathsf{edge}(q_1) \wedge \cdots \wedge \exists^{\geq m_t}\mathsf{edge}(q_t)$$

`function cons`(u: node, w: $\mathcal{L}_{ext}$-constraint): `boolean`
{ `if` $t = 0$ `then return` *true*;
`if` $t > 0$ `and` $\mathrm{OutEdges}(u, S) = \emptyset$ `then return` *false*;
`for` $K = 1$ `to` $\sum_{i=1}^{t} m_i$
`if` $\mathcal{T} \models \exists x_1 \cdots \exists x_K (F \wedge G \wedge H)$ `then return` *true*;
`return` *false*;
}

where F, G, and H are the following formulae of $\mathcal{T}$:

$$F = \bigwedge_{1 \leq i \leq t} \bigvee_{1 \leq j_1 < \cdots < j_{m_i} \leq K} \bigwedge_{1 \leq k \leq m_i} q_i(x_{j_k})$$

$$G = \bigwedge_{1 \leq i \leq s} \bigwedge_{1 \leq j_1 < \cdots < j_{n_i+1} \leq K} \neg \bigwedge_{1 \leq k \leq n_i+1} p_i(x_{j_k})$$

$$H = \bigwedge_{1 \leq i \leq K} \bigvee_{1 \leq j \leq \ell} r_j(x_i)$$

Theorem 1. *Given a node u of an $\mathcal{L}_{ext}$-schema S, we have that u is consistent iff* $\mathtt{cons}(u, \mathcal{C}(u))$ *returns true. Moreover,* $\mathtt{cons}(u, \mathcal{C}(u))$ *runs in time polynomial in the number of conjuncts in $\mathcal{C}(u)$, and in time exponential in the numeric indexes in $\mathcal{C}(u)$.*

By exploiting `cons` we can construct the function `rinext`, which removes the inconsistent nodes from an $\mathcal{L}_{ext}$-schema as follows:

`function rinext`(S: $\mathcal{L}_{ext}$-schema): $\mathcal{L}_{ext}$-schema
{ `repeat`
  `if` there is a node $u \in \text{Nodes}(S)$ that satisfies one of the following conditions:
  - u is not reachable with a direct path starting from root(S);
  - `cons`$(u, \mathcal{C}(u)) = \mathit{false}$

  `then` remove from S, u, all the edges outgoing from u, and all the edges incoming in u;
`until` root(S) is removed from S `or` no new node has been removed from S;
`return` S
}

Theorem 2. *Given an $\mathcal{L}_{ext}$-schema S, we have that S is consistent iff* `rinext`(S) *does not remove* root(S). *Moreover,* `rinext`(S) *runs in time polynomial in the number of nodes of S, and in time exponential in the numeric indexes of the constraints labeling the nodes of S.*

Theorem 2 provides a bound on schema consistency which is polynomial in the number of nodes, but exponential in the size of the schema. We can also show that $\mathcal{L}_{ext}$-schema consistency can be a checked by a nondeterministic algorithm which runs in polynomial time in the size of the schema.

Theorem 3. *Given an $\mathcal{L}_{ext}$-schema S, verifying the consistency of a node u of S can be done in nondeterministic polynomial time in the size of $\mathcal{C}(u)$.*

3.2 Propositional Local Constraints

We further extend the expressive power of extended local constraints by adding disjunction to $\mathcal{L}_{ext}$; we thus obtain a language endowed with all the operators of propositional logics (including negation, since, e.g., $\exists^{\leq n}\mathsf{edge}(p)$ is equivalent to $\neg\exists^{\geq n+1}\mathsf{edge}(p)$). The language $\mathcal{L}_{prop}$ of *propositional local constraints* is defined by the following abstract syntax:

$$\gamma \ ::= \ \top \ \mid \ \exists^{\leq n}\mathsf{edge}(p) \ \mid \ \exists^{\geq n}\mathsf{edge}(p) \ \mid \ \gamma_1 \wedge \gamma_2 \ \mid \ \gamma_1 \vee \gamma_2$$

where p is a unary formula of $\mathcal{T}$ and $n \in \mathbb{N}$. The semantics of the new operator we introduced is as follows:

Syntax	**Semantics**
$v \rightarrowtail \gamma_1 \vee \gamma_2$	$(v \rightarrowtail \gamma_1) \vee (v \rightarrowtail \gamma_2)$

Example 4. Let us consider again the schema of Figure 1. We would like to enforce, in addition to the properties specified in Example 3, also the property (5) each student who has given less than 5 exams must have a tutor. This can be done by using constraints of $\mathcal{L}_{prop}$ as follows:

$\mathcal{C}(u_1) = \mathcal{C}(u_5) = \mathcal{C}(u_6) = \top$ $\qquad$ $\mathcal{C}(u_2) = \exists^{\geq 1}\mathsf{edge}(\text{Student}) \wedge \exists^{\geq 1}\mathsf{edge}(\text{Department})$
$\mathcal{C}(u_4) = \exists^{\geq 1}\mathsf{edge}(\text{Professor})$ $\qquad$ $\mathcal{C}(u_3) = \exists^{\geq 5}\mathsf{edge}(\text{Exam}) \vee (\exists^{\geq 1}\mathsf{edge}(\text{Tutor}) \wedge \exists^{\leq 1}\mathsf{edge}(\text{Tutor}))$

We will see that there are significant advantages if we put the constraints of $\mathcal{L}_{prop}$ in *disjunctive normal form* (DNF), i.e., given a node u, we have

$$\mathcal{C}(u) = C_1 \vee \cdots \vee C_\nu$$

where C_k, with $1 \leq k \leq \nu$, are *clauses* of the form

$$C_k \;=\; \exists^{\leq n_1}\mathsf{edge}(p_1) \wedge \cdots \wedge \exists^{\leq n_s}\mathsf{edge}(p_s) \wedge \exists^{\geq m_1}\mathsf{edge}(q_1) \wedge \cdots \wedge \exists^{\geq m_t}\mathsf{edge}(q_t)$$

Exploiting the fact that the constraints are in DNF, and that each clause of $\mathcal{C}(u)$ is a constraint of $\mathcal{L}_{ext}$, we can use the function `cons`, which checks the consistency of nodes labeled with constraints of $\mathcal{L}_{ext}$, to check the consistency of nodes of $\mathcal{L}_{prop}$-schemas. Notice that for $\mathcal{L}_{prop}$ we *cannot* remove the negative constraints as we did for $\mathcal{L}_{sl}$ and $\mathcal{L}_{ext}$. Instead, we can remove the negative constraints *clause by clause*, as each clause of $\mathcal{L}_{prop}$ is a constraint of $\mathcal{L}_{ext}$. So, let `rnec` be the function that removes the negative constraints from a clause.

The function which checks the consistency of an $\mathcal{L}_{prop}$-schema by means of local checks is the following, hence showing that $\mathcal{L}_{prop}$ is a local constraint language.

`function rinprop`(S: schema): schema
{ `repeat`
  `if` there is a node u in S with $\mathcal{C}(u) = C_1 \vee \cdots \vee C_\nu$,
    such that for each $K \in \{1, \ldots, \nu\}$, at least one of the following conditions is verified:
    - u is not reachable starting from root(S) through a direct path;
    - $t \geq 1$ and `cons`$(u, $`rnec`$(C_K)) = \mathit{false}$
      where `rnec`$(S, C_K) = \exists^{\leq n_1}\mathsf{edge}(p_1) \wedge \cdots \wedge \exists^{\leq n_s}\mathsf{edge}(p_s) \wedge \exists^{\geq m_1}\mathsf{edge}(q_1) \wedge \cdots \wedge \exists^{\geq m_t}\mathsf{edge}(q_t)$,
  `then` remove from S, u, all the edges outgoing from u, and all the edges incoming in u;
  `until` root(S) is removed from S `or` no new node has been removed from S;
  `return` S;
}

As for `rinext`, it can be shown that `rinprop`(S) runs in time polynomial in the number of nodes of S, and in time exponential in the numeric indexes of the constraints labeling the nodes of S.

We can prove that also for $\mathcal{L}_{prop}$-schemas, the problem of checking node consistency is in NP. The proof is very similar to that of Theorem 3.

3.3 Subsumption

We now discuss a general technique to check schemas for subsumption which can be applied whenever the constraint language used in the schemas is local. Given two schemas S and S', for each pair of nodes u of S and u of S' we try to build a fragment of database conforming to S_u and not conforming to $S'_{u'}$ (as usual, we neglect the constraints over the successors of u and u'). If we can build such a fragment, we remove the pair of nodes. At the end, if $(\text{root}(S), \text{root}(S'))$ is removed, we can in any case fix the fragments together to form a database which conforms to S and does not conform to S'; this database is a counterexample for the subsumption between S and S', and its existence proves that S is not subsumed by S'. On the contrary, if $(\text{root}(S), \text{root}(S'))$ is

not removed, we can deduce that $S \sqsubseteq S'$, since the relation from Nodes(S) to Nodes(S') that we have constructed can be used to extend every simulation from a database g to S to a simulation from g to S'.

In order to make the local check described above, we can call the function `cons`, giving to it a suitable combination of constraints as second argument. To check whether there exists a fragment respecting $\mathcal{C}(u)$ that violates an atomic constraint φ, which is part of $\mathcal{C}(u')$, we call $\texttt{cons}(u, \mathcal{C}(u) \wedge \neg\varphi)$.

With this technique we can define two functions `subsext` and `subsprop`, which check the subsumption between two $\mathcal{L}_{ext}$-schemas and two $\mathcal{L}_{prop}$-schemas respectively. The two functions are very similar; in particular, `subsprop` exploits the property that the $\mathcal{L}_{prop}$-constraints are in DNF, making *clause by clause* the same checks that `subsext` makes. The two functions make a number of calls to `cons` which is polynomial in the total number of nodes of the two schemas. Hence $\texttt{subsext}(S_1, S_2)$ and $\texttt{subsext}(S_1, S_2)$ run in time polynomial in the total number of nodes of S_1 and S_2.

The only exponential dependence of the execution time of $\texttt{subsext}(S_1, S_2)$ and $\texttt{subsext}(S_1, S_2)$ is that from the numeric indexes of the constraints of the nodes of S_1 and S_2. This is due to the fact that the function `cons` builds formulae to be checked for validity, whose size is exponential in the numeric indexes.

Theorem 4. *Checking subsumption between two $\mathcal{L}_{ext}$-schemas or two $\mathcal{L}_{prop}$-schemas S and S' can be done in time polynomial in the total number of nodes of S and S', and in time exponential in the numeric indexes of the constraints labeling the nodes of the schemas.*

4 Bidirectional Schemas

In the framework we have adopted till now (both with and without constraints) there is an evident asymmetry in schemas between incoming and outgoing edges: both the notion of simulation and the constraint languages we have considered essentially talk about outgoing edges only. We investigate now the extension of the framework obtained by removing such an asymmetry, and considering both outgoing and incoming edges in the same way. The ability to refer to links in both directions substantially increases the expressive power of a representation formalism, and has been considered important in traditional database models [11, 10], in knowledge representation formalisms [14, 10], and in models and query languages for semistructured data [12, 19, 9]. Notably, XLink [19], the linking language of XML, allows one to express bidirectional and backward links.

We consider so called *bidirectional* databases and schemas, which, as before, are edge-labeled graphs with one distinguished node called *root*. Since we want to maintain a perfect symmetry between edges traversed in both directions, we require that in bidirectional databases and schemas all nodes be connected to the root by a *semipath* (instead of a path).

First of all we extend the notion of simulation to take into account both outgoing and incoming edges.

Definition 6. *A* bidirectional simulation *from a (bidirectional) database g to a (bidirectional) schema S is a binary relation $\trianglelefteq$ from* Nodes(g) *to* Nodes(S) *such that, for each pair of nodes v in g and u in S with $v \trianglelefteq u$ the following hold:*

1. *For each edge* $v \xrightarrow{c} v_s$ *in* g *there exists an edge* $u \xrightarrow{\pi} u_s$ *in* S *such that* $\mathcal{T} \models \pi(c)$ *and* $v_s \trianglelefteq u_s$.
2. *For each edge* $v_p \xrightarrow{c} v$ *in* g *there exists an edge* $u_p \xrightarrow{\pi} u$ *in* S *such that* $\mathcal{T} \models \pi(c)$ *and* $v_p \trianglelefteq u_p$.

The notions of *conformance*, *subsumption*, and *equivalence* extend straightforwardly to bidirectional databases and schemas, by considering bidirectional simulations instead of (unidirectional) simulations. Interestingly, also the polynomial algorithms for conformance and subsumption given in [5] can be extended to bidirectional databases and schemas while maintaining the same complexity bounds.

Theorem 5. *Checking subsumption between two bidirectional schemas* S *and* S' *can be done in polynomial time in the size of* S *and* S'.

4.1 Bidirectional Schemas with Constraints

We consider now the extension of bidirectional schemas with constraints, by investigating both languages whose constraints are on the outgoing edges only, and languages whose constraints are on outgoing and incoming edges. Independently of the constraint language we consider, we can extend simulations to take into account both constraints and incoming edges. More precisely, for bidirectional schemas with constraints, the notion of *simulation* is defined by combining the condition on constraints in Definitions 2 with the conditions on both incoming and outgoing edges in Definition 6. Also, the notions of *conformance*, *subsumption*, *equivalence*, and *consistency* extend straightforwardly to bidirectional schemas with constraints.

It is interesting to observe that, when we consider a constraint language that is local and allows one to express constraints only on the outgoing edges, all techniques developed for inference on unidirectional schemas with constraints can be extended to take into account bidirectional databases and schemas (and hence bidirectional simulations). In particular, this holds both for the language $\mathcal{L}_{sl}$ of simple local constraints introduced in Section 2.2 and for the more expressive constraint languages $\mathcal{L}_{ext}$ and $\mathcal{L}_{prop}$ discussed in Section 3.

Theorem 6. *Checking subsumption between two bidirectional* $\mathcal{L}_{sl}$*-schemas* S *and* S' *can be done in polynomial time in the size of* S *and* S'.

Theorem 7. *Checking subsumption between two bidirectional* $\mathcal{L}_{ext}$*-schemas or* $\mathcal{L}_{prop}$*-schemas* S *and* S' *can be done in time polynomial in the number of nodes of* S *and* S'*, and exponential in the numeric indexes (represented in unary) of the constraints labeling the nodes of* S *and* S'.

4.2 Bidirectional Schemas with Bidirectional Constraints

Until now we have considered only schemas and constraint languages for which the constraints are imposed only on the outgoing edges of a node. In the context of bidirectional databases and schemas, it is quite natural to allow one to impose constraints also on the incoming edges of a node. In other words, we are allowed to express *bidirectional constraints* on the schema. For example, using such types of constraints one may require that a page representing a subsection of a document is referenced in at least one page representing a section of a document.

Obviously, imposing constraints on incoming edges could also be done in unidirectional schemas. However, there is no natural justification for the asymmetry resulting from the fact that these

Syntax	Semantics
$u \rightarrowtail \top$	Always
$u \rightarrowtail \exists \mathsf{edge}(p)$	$\exists u \xrightarrow{c} u_s \in \text{Edges}(g) \mid \mathcal{T} \models p(c)$
$u \rightarrowtail \exists \mathsf{edge}^-(p)$	$\exists u_p \xrightarrow{c} u \in \text{Edges}(g) \mid \mathcal{T} \models p(c)$
$u \rightarrowtail \exists^{\leq 1} \mathsf{edge}(p)$	$\#\{u \xrightarrow{c} u_s \in \text{Edges}(g) \mid \mathcal{T} \models p(c)\} \leq 1$
$u \rightarrowtail \exists^{\leq 1} \mathsf{edge}^-(p)$	$\#\{u_p \xrightarrow{c} u \in \text{Edges}(g) \mid \mathcal{T} \models p(c)\} \leq 1$
$u \rightarrowtail \gamma_1 \wedge \gamma_2$	$(u \rightarrowtail \gamma_1) \wedge (u \rightarrowtail \gamma_2)$

Table 3. Syntax and semantics of $\mathcal{L}_{bid}$

constraints do not interact with the conditions imposed by the simulation. Moreover some types of constraints on incoming edges are trivially satisfied in unidirectional schemas by adding artificial nodes which are not reachable from the root by a direct path.

We show that, even in the case where the constraint language is quite simple, by allowing bidirectional constraints we lose several desirable properties of schemas (notably locality of constraints), and this has dramatic consequences on the complexity of consistency and subsumption.

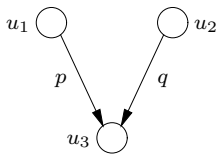
We consider the constraint language $\mathcal{L}_{bid}$, which generalizes $\mathcal{L}_{sl}$ with constraints on incoming edges:

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

where p is a formula of $\mathcal{T}$. Without loss of generality we have not included in $\mathcal{L}_{bid}$ constraints of the form $\nexists \mathsf{edge}(p)$ and $\nexists \mathsf{edge}^-(p)$, since we can remove such constraints as done for $\mathcal{L}_{sl}$. Constraints of the form $\exists \mathsf{edge}(p)$ and $\exists^{\leq 1} \mathsf{edge}(p)$ are called *direct constraints*, and constraints of the form $\exists \mathsf{edge}^-(p)$ and $\exists^{\leq 1} \mathsf{edge}^-(p)$ are called *inverse* constraints. The semantics of $\mathcal{L}_{bid}$ is given in Table 3.

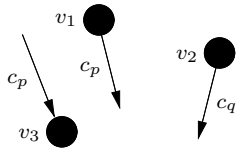
When only direct (or only inverse) constraints are present in an $\mathcal{L}_{bid}$-schema, there is no interaction between the edges of a database that need to be considered to verify whether different constraints are satisfied. Hence, the consistency of a node can be verified independently from the constraints present on the other nodes. This is no longer true when both types of constraints are present. Indeed, for an $\mathcal{L}_{bid}$-schema S, consider two nodes u_1 and u_2 with an edge $u_1 \xrightarrow{\pi} u_2$. If $\mathcal{C}(u_1)$ imposes constraints on outgoing edges and $\mathcal{C}(u_2)$ imposes constraints on incoming edges, it is obvious that, in a database simulating S, satisfaction of $\mathcal{C}(u_1)$ may in general depend on satisfaction of $\mathcal{C}(u_2)$.

Example 5. Consider the $\mathcal{L}_{bid}$-schema S shown in Figure 2(a), and suppose that in $\mathcal{T}$ the formulae $p(c_p)$, $q(c_q)$, and $\neg\exists x(p(x) \wedge q(x))$ are valid. If we consider each node of S separately, we can find for each one a set of constants that can be used to label edges of databases in such a way that the constraints (considered separately) are satisfied. In Figure 2(b) three such fragments are shown. However, in order to construct a database that conforms to S we have to merge the various fragments together. Due to the interaction of the constraints on outgoing edges (for u_1 and u_2) and incoming edges (for u_3) this is not possible without violating any constraint. This holds for the database fragments shown in Figure 2(b), but it is easy to see that in general it is not possible to find a database that conforms to S in which there are nodes simulating each of u_1, u_2, and u_3. Therefore, all three nodes of S are not consistent together, while they are consistent when considered separately.



$\mathcal{C}(u_1) = \exists \mathsf{edge}(p)$
$\mathcal{C}(u_2) = \exists \mathsf{edge}(q)$
$\mathcal{C}(u_3) = \exists \mathsf{edge}^-(p) \wedge \exists^{\leq 1} \mathsf{edge}^-(p \vee q)$

(a) Schema S



(b) Fragments of databases

Fig. 2. Schema in which bidirectional constraints interact, and fragments of databases

$\mathcal{C}(u) = \exists \mathsf{edge}(p) \wedge \exists \mathsf{edge}(\neg p) \wedge \exists^{\leq 1} \mathsf{edge}^-(true)$

Fig. 3. Schema which admits only conforming databases that are infinite

Example 5 shows that $\mathcal{L}_{bid}$ is indeed not local, and hence methods for checking consistency and subsumption based on local validity checks, such as those presented in Section 3, cannot be applied in this case. Indeed, the problem of reasoning on $\mathcal{L}_{bid}$ schemas is still open.

A further consequence of the expressiveness of $\mathcal{L}_{bid}$ is that for $\mathcal{L}_{bid}$-schemas the *finite model property* does not hold. This means that there is an $\mathcal{L}_{bid}$-schema such that every database conforming to it must have an infinite number of nodes, as shown by the following example.

Example 6. Consider a database g that conforms to the $\mathcal{L}_{bid}$-schema S shown in Figure 3. Since the root r of g has to simulate u, the constraints $\exists \mathsf{edge}(p)$ and $\exists \mathsf{edge}(\neg p)$ in $\mathcal{C}(u)$ impose that r has two outgoing edges to two nodes v_1 and v_2. Since v_1 and v_2 must again simulate u, they must in particular satisfy $\exists^{\leq 1} \mathsf{edge}^-(true)$, and hence v_1 and v_2 cannot coincide[3]. By applying the same argument as for r also to v_1 and v_2 one can see that g must necessarily contain an infinite number of nodes. One the other hand, the infinite database consisting of a binary tree in which each left edge is labeled with a constant satisfying p, and each right edge is labeled with a constant satisfying $\neg p$ conforms to S, showing that S is indeed consistent.

The lack of the finite model property is typical of representation formalisms that can express functionality on links in both directions, and in general makes it necessary to adopt different techniques for the cases where one wants to reason wrt finite databases only, or one wants to consider arbitrary (possibly infinite) databases [13, 6].

5 Conclusions

We have studied the relationship between various extensions of constraint languages for semistructured data schemas and the basic property of locality, which allows one to check subsumption between schemas in polynomial time in the number of nodes of the schemas. We have shown that locality holds when both numeric constraints and disjunction are added to a simple constraint language. On the other hand, locality is lost when we consider constraints both on outgoing and incoming edges of databases.

We would like to point out that, while the results in this paper have been established in the formal framework of the BDFS data model [5] based on the notion of simulation, all considerations relative to locality hold also for other data models, such as OEM [2].

[3] One of v_1 or v_2 could coincide with r, but not both.

References

1. S. Abiteboul. Querying semi-structured data. In *Proc. of ICDT'97*, pages 1–18, 1997.
2. 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.
3. J. R. Abrial. Data semantics. In *Data Base Management*, pages 1–59. North-Holland Publ. Co., Amsterdam, 1974.
4. A. Borgida, R. J. Brachman, D. L. McGuinness, and L. Alperin Resnick. CLASSIC: A structural data model for objects. In *Proc. of ACM SIGMOD*, pages 59–67, 1989.
5. P. Buneman, S. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT'97*, pages 336–350, 1997.
6. D. Calvanese. Finite model reasoning in description logics. In *Proc. of KR'96*, pages 292–303, 1996.
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. D. Calvanese, G. De Giacomo, and M. Lenzerini. Modeling and querying semi-structured data. *Network and Information Systems*, 2(2), 1999.
9. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of PODS 2000*, 2000.
10. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.
11. R. G. G. Cattell and D. K. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, Los Altos, 1997.
12. J. Clark and S. Deach. Extensible Stylesheet Language (XSL). Technical report, World Wide Web Consortium, 1999. Available at http://www.w3.org/TR/WD-xsl.
13. S. S. Cosmadakis, P. C. Kanellakis, and M. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37(1):15–46, 1990.
14. G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of AAAI'94*, pages 205–212, 1994.
15. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.
16. 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.
17. D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
18. J. Grant and J. Minker. Inferences for numerical dependencies. *Theor. Comp. Sci.*, 41:271–287, 1985.
19. E. Maler and S. DeRose. XML Linking Language (XLink) – W3C working draft 03-march-1998. Technical report, World Wide Web Consortium, 1998. Available at http://www.w3.org/TR/1998/WD-xlink-19980303.
20. A. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.
21. 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.
22. B. Thalheim. Fundamentals of cardinality constraints. In *Proc. of ER'92*, pages 7–23. Springer-Verlag, 1992.