Decidable Containment of Recursive Queries

Diego Calvanese[1], Giuseppe De Giacomo[1], and Moshe Y. Vardi[2]

[1] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
`lastname@dis.uniroma1.it`,
`http://www.dis.uniroma1.it/~lastname/`

[2] Department of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
`vardi@cs.rice.edu`
`http://www.cs.rice.edu/~vardi/`

Abstract. One of the most important reasoning tasks on queries is checking containment, i.e., verifying whether one query yields necessarily a subset of the result of another one. Query containment, is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering. Containment is undecidable in general for Datalog, the fundamental language for expressing recursive queries. On the other hand, it is known that containment between monadic Datalog queries and between Datalog queries and unions of conjunctive queries are decidable. It is also known that containment between unions of conjunctive two-way regular path queries (UC2RPQs), which are queries used in the context of semistructured data models containing a limited form of recursion in the form of transitive closure, is decidable. In this paper we combine the automata-theoretic techniques at the base of these two decidability results to show that containment of Datalog in UC2RPQs is decidable in 2EXPTIME.

1 Introduction

Querying is the fundamental mechanism for extracting information from a database. The basic reasoning task associated to querying is query answering, which amounts to computing the information to be returned as result of a query. There are, however, other reasoning services involving queries that data and knowledge representation systems should support. One of the most important is checking containment, i.e., verifying whether one query yields necessarily a subset of the result of another one. Query containment, called *subsumption* in AI [1,2], is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering; cf. [3,4,5,6,7,8,9,10,11,12,13]. Thus, it is fair to describe query containment as one of the most fundamental database reasoning tasks.

Needless to say, query containment is undecidable if we do not limit the expressive power of the query language; it is clearly undecidable for first-order logic. In fact, in knowledge representation suitable query languages have been designed for retaining decidability. The same is true in databases, where the notion of *conjunctive query* is the basic one in the investigation of reasoning about queries [14]. A conjunctive query (CQ) is simply a conjunction of atoms, where each atom is built out from relation symbols and (existentially quantified) variables. Relationally, a CQ is a project-join query. By adding union and recursion to conjunctive queries, one gets *Datalog*, the language of logic programs (known also as Horn-clause programs) without function symbols [15], which is essentially a fragment of fixpoint logic [16,17]. Datalog consists, in a pure way, only of the most fundamental elements of relational queries: join, projection, union, and recursion. With respect to query containment, CQs and Datalog span the spectrum in terms of computational complexity. In [14] it is shown that CQ containment is equivalent to CQ evaluation (NP-complete). (For some extensions, see [18,19,20,21].) On the other hand, it is shown in [22] that containment of Datalog queries is undecidable; the proof is by reduction from the containment problem for context-free grammars.

The most powerful query-containment results for Datalog are given in [23,24, 25]. In [23] it is pointed out that tree-automata techniques can be used to prove the decidability of query containment for *monadic* Datalog, where rule heads use a single variable (which means that intermediate result of the query, as well as the final one, are sets of data elements). The other results apply to the relationship between Datalog and non-recursive Datalog (non-recursive Datalog queries are in essence unions of conjunctive queries). In [24] it is shown that checking containment of nonrecursive Datalog queries in Datalog queries is decidable in exponential time. In [25] (see also [21]) it is shown, using tree-automata techniques, that containment of Datalog queries in nonrecursive Datalog queries is decidable in triply exponential time. When the non-recursive query is represented, via unfolding, as a union of CQs, the complexity is doubly exponential, rather than triple exponential. (These bounds are known to be optimal, see [26, 4] for studies of special cases and some extensions.)

In this paper we address the problem of query containment in the context of semistructured data models. Our goal is to capture the essential features found in databases, both traditional and semistructured, as well as knowledge bases in semantic networks, conceptual graphs, and description logics. For this purpose, we conceive a database as an edge-labeled graph, where nodes represent objects, and a labeled edge between two nodes represents the fact that the binary relation denoted by the label holds for the objects. This model captures data expressed using XML-like languages [27,28] and is accepted as a standard model for semistructured data [29,30].

In this framework, a basic querying mechanism is the one of *regular path queries* (RPQ) [29,31,32], which ask for all pairs of objects that are connected by a path conforming to a regular expression. Regular path queries are extremely useful for expressing complex navigations in a graph. In particular, union and transitive closure are crucial when we do not have a complete knowledge of the structure of the database. In our regular path queries, we include also the *inverse* operator, which enables us to navigate edges backwards [29,7], for example, from a child to its parent. We denote these queries by 2RPQs (two-way regular path queries). Using 2RPQs as the basic querying mechanism, one can construct *conjunctive 2-way regular path queries* (C2RPQs), which enables us to perform joins and projections over 2RPQs. C2RPQs are the basic building blocks for querying semistructured data [33,13,31]. The containment problem for C2RPQs (actually for UC2RPQs, unions of such C2RPQs) was studied in [34] (see also [33]), where it was shown, using two-way automata, to be EXPSPACE-complete.

The notable fact about the decidability of containment for C2RPQs is that C2RPQs are a fragment of recursive Datalog, due to the transitive closure operator. Thus, the result in [33,34] is the first decidability result for containment of non-monadic recursive Datalog queries. The fact that automata-theoretic techniques are used both in [25] and in [34] suggests that perhaps the two decidability results can be combined. We show here that this is indeed the case by proving the decidability of the containment of Datalog queries in UC2RPQs (which, implies the known decidability result for containment of UC2RPQs). The automata-theoretic techniques combine tree automata with two-way automata; we use alternating two-way tree automata [35]. The upper bound is doubly exponential time, just as in [25], which we conjecture is optimal (see Conclusions).

2 Databases and Queries

We consider a *semistructured database* (DB) $\mathcal{G}$ as an edge-labeled graph $(\mathcal{D}, \mathcal{E})$, where $\mathcal{D}$ is the set of nodes, and $\mathcal{E}$ is the set of edges labeled with elements of an alphabet Δ. A node represents an object, and an edge between nodes d_1 and d_2 labeled e, denoted $e(d_1, d_2)$, represents the fact that the binary relation e holds for the pair (d_1, d_2).

The basic querying mechanism on a DB is that of *regular path queries* (RPQs). An RPQ E is expressed as a regular expression or a finite automaton, and computes the set of pairs of nodes of the DB connected by a path that conforms to the regular language $L(E)$ defined by E. We consider unions of conjunctive 2-way regular path queries (UC2RPQs) [34], which extend regular path queries with the possibility to traverse edges backwards, with conjunctions and variables, and with union.

Formally, Let Δ be a set of binary relation symbols, and let $\Delta^{\pm} = \Delta \cup \Delta^{-}$, with $\Delta^{-} = \{e^{-} \mid e \in \Delta\}$. Intuitively, e^{-} denotes the inverse of the binary relation e. If $r \in \Delta^{\pm}$, then we use r^{-} to mean the *inverse* of the relation r, i.e., if r is e, then r^{-} is e^{-}, and if r is e^{-}, then r^{-} is e.

2-way regular path queries (2RPQs) are expressed by means of regular expressions or finite word automata over $\Delta^{\pm}$. Thus, in contrast with RPQs, 2RPQs may use also the inverse e^{-} of e, for each $e \in \Delta$. When evaluated over a DB $\mathcal{G}$, a 2RPQ E computes the set $E(\mathcal{G})$ of pairs of nodes (d_0, d_q) such that $r_1(d_0, d_1), r_2(d_1, d_2), \ldots, r_q(d_{q-1}, d_q)$ hold in $\mathcal{G}$ and $r_1 r_2 \cdots r_q$ is in the regular language $L(E)$ defined by E. Observe that, when $q = 0$, we have that $r_1 r_2 \cdots r_q = \varepsilon$ and $d_0 = d_q$.

Conjunctive 2-way regular path queries (C2RPQs) are conjunctions of atoms, where each atom specifies that one 2RPQ holds between two variables. More precisely a C2RPQ γ of *arity* n is a formula of the form

$$Q(x_1, \ldots, x_n) \;\leftarrow\; E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$$

where $x_1, \ldots, x_n, y_1, y_1' \ldots, y_m, y_m'$ range over a set $\{u_1, ..., u_k\}$ of variables, each x_i, called a *distinguished variable*, is one of $y_1, y_1' \ldots, y_m, y_m'$, and $E_1, \ldots, E_m$ are 2RPQs. The *answer set* $\gamma(\mathcal{G})$ to a C2RPQ γ over a DB $\mathcal{G} = (\mathcal{D}, \mathcal{E})$ is the set of tuples $(d_1, \ldots, d_n)$ of nodes of $\mathcal{G}$ such that there is a total mapping σ from $\{u_1, \ldots, u_k\}$ to $\mathcal{D}$ with $\sigma(x_i) = d_i$ for every distinguished variable x_i of γ, and $(\sigma(y), \sigma(y')) \in E(\mathcal{G})$ for every conjunct $E(y, y')$ in γ.

Finally, a *union of conjunctive 2-way regular path queries* (UC2RPQ) of arity n has the form $\cup_i \gamma_i$, where each γ_i is a C2RPQ of arity n. The answer set to a UC2RPQ $\Gamma = \cup_i \gamma_i$ over a DB $\mathcal{G}$ is simply $\Gamma(\mathcal{G}) = \cup_i \gamma_i(\mathcal{G})$. Notice that traditional *conjunctive queries* (resp., unions of conjunctive queries) (cf. [15]) are just a special case of C2RPQs (resp., UC2RPQ) in which each 2RPQ in an atom is simply a relation symbol.

A Datalog program consists of a set of Horn rules. A *(Horn) rule* is a first order material implication between a head and a body, where the head consists of a single atom, and the body consists of a conjunction of atoms. Each atom is a formula of the form $R(x_1, \ldots, x_n)$ where R is a predicate symbol and $x_1, \ldots, x_n$ are variables. All variables are implicitly universally quantified outside the rule. The predicates that occur in heads of rules are called *intensional* (IDB) predicates. The rest of the predicates are called *extensional* (EDB) predicates. Since we consider Datalog programs that are evaluated over a semistructured database, the EDB predicates have to be among the predicates in Δ, which are all binary. Observe, however, that IDB predicates, which are not in Δ, may be of arbitrary arity.

Let Π be a Datalog program. Let $Q^i_\Pi(\mathcal{G})$ be the collection of facts about an IDB predicate Q that can be deduced from a database $\mathcal{G}$ by at most i applications of the rules in Π, and let $Q^\infty_\Pi(\mathcal{G})$ be the collection of facts about Q that can be deduced from $\mathcal{G}$ by any number of applications of the rules in Π, that is,

$$Q^\infty_\Pi(\mathcal{G}) = \bigcup_{i \geq 0} Q^i_\Pi(\mathcal{G})$$

We say that a Datalog program Π with goal predicate Q is *contained* in a UC2RPQ Γ if $Q^\infty_\Pi(\mathcal{G}) \subseteq \Gamma(\mathcal{G})$ for every database $\mathcal{G}$.

3 Containment of Datalog in Unions of Conjunctive Queries

A *containment mapping* from a conjunctive query ψ to a conjunctive query φ is a renaming of variables subject to the following constraints: (a) every distinguished variable must map to itself, and (b) after renaming, every literal in ψ must be among the literals of φ. It is well known that containment of conjunctive queries can be characterized in terms of containment mappings (cf. [15]). In fact this characterization has been extended in [19] to unions of conjunctive queries, and holds also for infinite unions.

Theorem 1 ([19]). *Let $\Phi = \cup_i \varphi_i$ and $\Psi = \cup_i \psi_i$ be (possibly infinite) unions of conjunctive queries. Then Φ is contained in Ψ (i.e., $\Phi(\mathcal{G}) \subseteq \Psi(\mathcal{G})$ for every database $\mathcal{G}$) if and only if each φ_i is contained in some ψ_j , i.e., there is a containment mapping from ψ_j to φ_i.*

As for containment of Datalog in (unions) of conjunctive queries, it is known (cf. [36,37]) that the relation defined by an IDB predicate in a Datalog program Π, i.e., $Q^\infty_\Pi(\mathcal{G})$, can be defined by an *infinite* union of conjunctive queries. That is, for each IDB predicate Q there is an infinite sequence $\varphi_0, \varphi_1, \ldots$ of conjunctive queries such that, for every database $\mathcal{G}$, we have $Q^\infty_\Pi(\mathcal{G}) = \bigcup_{i=0}^\infty \varphi_i(\mathcal{G})$. The φ_i's are called the *expansions* of Q. In [25], expansions of a Datalog program Π are described in terms of so-called *expansion trees*, in which each node is labeled with an instance of a rule of Π. We call head and body of a node the head and the body of the rule labeling the node, respectively. In an expansion tree for an IDB predicate Q, the root is labeled by a rule whose head is a Q-atom. If a node g is labeled by a rule instance

$$R(\mathbf{t}) \leftarrow R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

where the IDB atoms in the body of the rule are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$, then g has children $g1, \ldots, g\ell$ labeled with rule instances whose heads are respectively the atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$. In particular, if all atoms in the body of g are EDB atoms, then g must be a leaf. The query corresponding to an expansion tree is the conjunction of all EDB atoms in the nodes of the tree, with the variables in the head of the root as the free variables. Thus, we can view an expansion tree τ as a conjunctive query. Let *trees*(Q, Π) denote the set of expansion trees for an IDB predicate Q in Π. (Note that *trees*(Q, Π) is an infinite set.) Then for every database $\mathcal{G}$, we have

$$Q^\infty_\Pi(\mathcal{G}) = \bigcup_{\tau \in \mathit{trees}(Q,\Pi)} \tau(\mathcal{G})$$

It follows that Π is contained in a conjunctive query φ if there is a containment mapping from φ to each expansion tree τ in *trees*(Q, Π), i.e., a mapping, which maps distinguished variables to distinguished variables and maps the atoms of φ to atoms in the bodies of rules labeling nodes of τ.

Unfortunately, the number of variables, and hence the number of node labels in expansion trees is not bounded, and thus expansion trees are not directly suited for an automata-theoretic approach to containment. In [25], the notion of *proof tree* is introduced, with the idea of describing expansion trees using a finite number of labels. The number of labels is bound by bounding the set of variables that can occur in labels of nodes in the tree. If r is a rule of a Datalog program Π, then let $num\_var(r)$ be the number of variables occurring in IDB atoms in r (head or body). Let $num\_var(\Pi)$ be twice the maximum of $num\_var(r)$ for all rules r in Π. Let $var(\Pi)$ be the set $\{x_1, \ldots, x_{num\_var(\Pi)}\}$. A *proof tree* for Π is simply an expansion tree for Π all of whose variables are from $var(\Pi)$. We denote the set of proof trees for a predicate Q of a Datalog program Π by $p\_trees(Q, \Pi)$.

A proof tree represents an expansion tree where variables are re-used. In other words, the same variable is used to represent a set of distinct variables in the expansion tree. Intuitively, to reconstruct an expansion tree for a given proof tree, we need to distinguish among occurrences of variables. Let g_1 and g_2 be nodes in a proof tree τ, with a lowest common ancestor g_0, and let x_1 and x_2 be occurrences, in g_1 and g_2, respectively, of a variable x. We say that x_1 and x_2 are *connected* in τ if the head of every node, except perhaps for g_0, on the simple path connecting g_1 and g_2 has an occurrence of x. We say that an occurrence x of a variable x in τ is a *distinguished occurrence* if it is connected to an occurrence of x in the head of the root of τ.

We want to define containment mappings from conjunctive queries to proof trees such that there is a containment mapping from a conjunctive query to a proof tree if and only if there is a containment mapping from the conjunctive query to the expansion corresponding to the proof tree. To do so, we need to force a variable in the conjunctive query to map to a unique variable in the expansion corresponding to the proof tree. A *strong containment mapping* from a conjunctive query φ to a proof tree τ is a containment mapping h from φ to τ with the following properties:

– h maps distinguished occurrences in φ to distinguished occurrences in τ, and
– if x_1 and x_2 are two occurrences of a variable x in φ, then the occurrences $h(\mathsf{x}_1)$ and $h(\mathsf{x}_2)$ in τ are connected.

The following characterization of containment of a union of conjunctive queries in a Datalog program was shown in [25].

Theorem 2 ([25]). *Let Π be a Datalog program with goal predicate Q, and let $\Phi = \cup_i \varphi_i$ be a (possibly infinite) union of conjunctive queries. Then Π is contained in Φ if and only if for every proof tree $\tau \in p\_trees(Q, \Pi)$ there is a strong containment mapping from some φ_i to τ.*

The above theorem is shown in [25] for *finite* unions of conjunctive queries only. However, it is easy to see that the proof carries through also for infinite unions.

Notice that Theorem 2 by itself does not provide decidability of containment of Datalog in (possibly infinite) unions of conjunctive queries, since one needs a method to check the existence of a strong containment mapping. Undecidability of containment between Datalog queries [22] shows that such a method will not exist in general for (infinite) unions that are expansions of Datalog programs. However, in [25] the above result is exploited to show that containment of a Datalog query in a finite union of conjunctive queries is in 2EXPTIME (and in fact 2EXPTIME-complete).

To exploit Theorem 2 for containment of Datalog queries in UC2RPQs, we need to characterize the problem in terms of containment between Datalog and (infinite) unions of conjunctive queries. An *expansion* of a C2RPQ

$$Q(x_1,\ldots,x_n) \ \leftarrow\ E_1(y_1,y_1'),\, \ldots,\, E_m(y_m,y_m')$$

is a CQ of the form

$$\begin{aligned} Q(x_1,\ldots,x_n) \ \leftarrow\ & r_1^1(y_1,z_1^1),\, r_1^2(z_1^1,z_1^2),\, \ldots,\, r_1^{n_1}(z_1^{n_1-1},y_1'),\\ & \quad\vdots\\ & r_m^1(y_m,z_m^1),\, r_m^2(z_m^1,z_m^2),\, \ldots,\, r_m^{n_m}(z_m^{n_m-1},y_m') \end{aligned}$$

where, for each $i \in \{1,\ldots,m\}$, we have that $n_i \geq 0$, that $r_i^1 \cdots r_i^{n_i} \in L(E_i)$, and that all variables z_i^j are pairwise distinct. Observe that, when $n_i = 0$, we have that $r_i^1 \cdots r_i^{n_i} = \varepsilon$, and $r_i^1(y_i,z_i^1),\, r_i^2(z_i^1,z_i^2),\, \ldots,\, r_i^{n_i}(z_i^{n_i-1},y_i')$ becomes simply $y_i = y_i'$. Notice that, due to transitive closure, a C2RPQ has in general an infinite number of expansions.

The following lemma is an easy consequence of Theorem 2 and of the semantics of UC2RPQs.

Lemma 1. *Let Π be a Datalog program with goal predicate Q, and let $\Gamma = \cup_i \gamma_i$ be a finite union of C2RPQs γ_i. Then Π is contained in Γ if and only if for every proof tree $\tau \in p\_trees(Q,\Pi)$ there is a γ_i and an expansion φ of γ_i such that there is a strong containment mapping from φ to τ.*

We show how to check this condition using tree automata.

4 Two-Way Alternating Tree Automata

We present the basic notions on automata used in the rest of the paper. We assume familiarity with the standard notions of (one-way) word automata (1NFAs) and (one-way) nondeterministic tree automata (1NTAs), and concentrate on two-way alternating tree automata (2ATAs).

Trees are represented as prefix closed finite sets of words over $\mathbb{N}$ (the set of positive natural numbers). Formally, a *tree* T is a finite subset of $\mathbb{N}$, such that if $g{\cdot}c \in T$, where $g \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $g \in T$ and if $c > 1$ then also $g{\cdot}(c-1) \in T$. The elements of T are called *nodes*, and for every $g \in T$, the nodes $g{\cdot}c \in T$, with $c \in \mathbb{N}$, are the *successors* of g. By convention we take $g{\cdot}0 = g$, and $g{\cdot}c{\cdot}(-1) = g$. By definition, the empty sequence ε is a member of every tree, and is called the *root*. Note that $\varepsilon \cdot -1$ is undefined. The *branching degree* $d(g)$

of a node g denotes the number of successors of g. If the branching degree of all nodes of a tree is bounded by k, we say that the tree has branching degree k. Given a finite alphabet Σ, a *Σ-labeled tree* τ is a pair (T, V), where T is a tree and $V : T \to \Sigma$ maps each node of T to an element of Σ. Σ-labeled trees are often referred to as *trees*, and if $\tau = (T, V)$ is a (labeled) tree and g is a node of T, we use $\tau(g)$ to denote $V(g)$.

Two-way alternating tree automata (2ATAs) [35,23], are a generalization of standard nondeterministic top-down tree automata (1NTAs) [38,39]) with both upward moves and with alternation. Let $\mathcal{B}(I)$ be the set of positive Boolean formulae over I, built inductively by applying $\wedge$ and $\vee$ starting from **true**, **false**, and elements of I. For a set $J \subseteq I$ and a formula $\varphi \in \mathcal{B}(I)$, we say that J *satisfies* φ if and only if, assigning **true** to the elements in J and **false** to those in $I \setminus J$, makes φ true. For a positive integer k, let $[k] = \{-1, 0, 1, \ldots, k\}$. A *two-way alternating tree automaton* (2ATA) over an alphabet Σ running over trees with branching degree k, is a tuple $\mathbf{A} = (\Sigma, S, \delta, s_0, F)$, where S is a finite set of states, $\delta : S \times \Sigma \to \mathcal{B}([k] \times S)$ is the transition function, $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of final states. The transition function maps a state $s \in S$ and an input letter $\sigma \in \Sigma$ to a positive Boolean formula over $[k] \times S$. Intuitively, if $\delta(s, \sigma) = \varphi$, then each pair (c, s') appearing in φ corresponds to a new copy of the automaton going to the direction suggested by c and starting in state s'.

A run ν of a 2ATA $\mathbf{A}$ over a labeled tree $\tau = (T, V)$ is a labeled tree (T_ν, V_ν) in which every node is labeled by an element of $T \times S$. A node f of T_ν labeled by (g, s) describes a copy of $\mathbf{A}$ that is in the state s and reads the node g of τ. The labels of adjacent nodes have to satisfy the transition function of $\mathbf{A}$. Formally, a run (T_ν, V_ν) is a $(T \times S)$-labeled tree satisfying:

1. $\varepsilon \in T_\nu$ and $V_\nu(\varepsilon) = (\varepsilon, s_0)$.
2. Let $f \in T_\nu$, with $V_\nu(f) = (g, s)$ and $\delta(s, V(g)) = \varphi$. Then there is a (possibly empty) set $C = \{(c_1, s_1), \ldots, (c_n, s_n)\} \subseteq [k] \times S$ such that:
   - C satisfies φ and
   - for all $i \in \{1, \ldots, n\}$, we have that $f \cdot i \in T_\nu$, $g \cdot c_i$ is defined, and $V_\nu(f \cdot i) = (g \cdot c_i, s_i)$.

A run $\nu = (T_\nu, V_n u)$ on a tree τ is *accepting* if, whenever a leaf of T_ν is labeled by (g, s), then $s \in F$. $\mathbf{A}$ *accepts* a labeled tree τ if it has an accepting run on τ. The set of trees accepted by $\mathbf{A}$ is denoted $\mathcal{T}(\mathbf{A})$. The *nonemptiness* problem for tree automata consists in deciding, given a tree automaton $\mathbf{A}$, whether $\mathcal{T}(\mathbf{A})$ is nonempty.

As shown in [23], 2ATAs can be converted to complementary 1NTAs with only a single exponential blowup. Moreover, it is straightforward to see that one can construct a 2ATA of polynomial size accepting the finite union of the languages accepted by n 2ATAs.

Proposition 1 ([23]). *Given a 2ATA $\mathbf{A}$ over an alphabet Σ, there is a 1NTA $\overline{\mathbf{A}}$ of size exponential in the size of $\mathbf{A}$ such that $\overline{\mathbf{A}}$ accepts a Σ-labeled tree τ if and only if τ is rejected by $\mathbf{A}$.*

Proposition 2. *Given n 2ATAs $\mathbf{A}_1, \ldots, \mathbf{A}_n$ over an alphabet Σ, there is a 2ATA $\mathbf{A}_\cup$ of size polynomial in the sum of the sizes of $\mathbf{A}_1, \ldots, \mathbf{A}_n$ such that $\mathcal{T}(\mathbf{A}_\cup) = \mathcal{T}(\mathbf{A}_1) \cup \cdots \cup \mathcal{T}(\mathbf{A}_n)$.*

We make also use of the following standard results for 1NTAs.

Proposition 3 ([40]). *Given 1NTAs $\mathbf{A}_1$ and $\mathbf{A}_2$ over an alphabet Σ, there is a 1NTA $\mathbf{A}_\cap$ of size polynomial in the size of $\mathbf{A}_1$ and $\mathbf{A}_2$ such that $\mathcal{T}(\mathbf{A}_\cap) = \mathcal{T}(\mathbf{A}_1) \cap \mathcal{T}(\mathbf{A}_2)$.*

Proposition 4 ([38,39]). *The nonemptiness problem for 1NTAs is decidable in polynomial time.*

5 Containment of Datalog in Unions of C2RPQs

The main feature of proof trees is the fact that the number of possible labels is finite; it is actually exponential in the size of Π. Because the set of labels is finite, the set of proof trees $p\_trees(Q, \Pi)$, for an IDB predicate Q in a program Π, can be described by a tree automaton.

Theorem 3 ([25]). *Let Π be a Datalog program with a goal predicate Q. Then there is a 1NTA $\mathbf{A}^{p\_trees}_{Q,\Pi}$, whose size is exponential in the size of Π, such that $\mathcal{T}(\mathbf{A}^{p\_trees}_{Q,\Pi}) = p\_trees(Q, \Pi)$.*

The automaton $\mathbf{A}^{p\_trees}_{Q,\Pi} = (\Sigma, \mathcal{I} \cup \{accept\}, \mathcal{I}_Q, \delta, \{accept\})$ defined in [25] is as follows. The state set $\mathcal{I}$ is the set of all IDB atoms with variables among $var(\Pi)$. The start-state set $\mathcal{I}_Q$ is the set of all atoms $Q(\mathbf{s})$, where the variables of $\mathbf{s}$ are in $var(\Pi)$. The alphabet is $\Sigma = \mathcal{I} \times \mathcal{R}$, where $\mathcal{R}$ is the set of instances of rules of Π over $var(\Pi)$. The transition function δ is constructed as follows. Let ϱ be the body of a rule instance in $\mathcal{R}$

$$R(\mathbf{t}) \;\leftarrow\; R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

– If the IDB atoms in ϱ are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_\ell}(\mathbf{t}^{i_\ell})$, then there is a transition[1]

$$\langle 1, R_{i_1}(\mathbf{t}^{i_1})\rangle \wedge \cdots \wedge \langle \ell, R_{i_\ell}(\mathbf{t}^{i_\ell})\rangle \;\in\; \delta(R(\mathbf{t}), (R(\mathbf{t}) \leftarrow \varrho))$$

– If all atoms in ϱ are EDB atoms, then there is a transition

$$\langle 0, accept\rangle \;\in\; \delta(R(\mathbf{t}), (R(\mathbf{t}) \leftarrow \varrho))$$

It is easy to see that the number of states and transitions in $\mathbf{A}^{p\_trees}_{Q,\Pi}$ is exponential in the size of Π.

We now show that strong containment of proof trees in a C2RPQ can be checked by tree automata as well. Let Π be a Datalog program with binary

[1] For uniformity, we use the notation of 2ATAs to denote the transitions of 1NTAs.

EDB predicates in Δ and with goal predicate Q, and let γ be a C2RPQ over $\Delta^{\pm}$ of the same arity as Q. We describe the construction of a 2ATA $\mathbf{A}^{\gamma}_{Q,\Pi}$ that accepts all proof trees τ in $p\_trees(Q,\Pi)$ such that there is an expansion φ of γ and a strong containment mapping from φ to τ.

We view γ as a set of atoms $E(x,y)$, where E is a 1NFA $E = (\Delta^{\pm}, S_E, s_E, \delta_E, f_E)$, with $s_E, f_E \in S_E$, and where, w.l.o.g., δ_E does not contain ε-transitions. Also, w.l.o.g., we assume that for two distinct atoms $E_1(x_1,y_1)$ and $E_2(x_2,y_2)$, E_1 and E_2 are distinct automata with disjoint sets of states, i.e., $S_{E_1} \cap S_{E_2} = \emptyset$. For a 1NFA E, we use E^f_s to denote the 1NFA identical to E, except that $s \in S_E$ and $f \in S_E$ are respectively the initial and final state of E^f_s.

Let V_γ be the set of variables appearing in the C2RPQ γ, and $V^+_\gamma = \{\bar{v}^1_E, \bar{v}^2_E \mid E(x,y) \in \gamma\}$, i.e., for each 1NFA $E(x,y) \in \gamma$, V^+_γ contains two special variables $\bar{v}^1_E$ and $\bar{v}^2_E$. We denote with $\mathcal{B}$ the set of all sets β of atoms, such that β contains, for each atom $E(x,y) \in \gamma$, at most one atom $E^f_s(x',y')$, for some $s, f \in S_E$, with x' either x or $\bar{v}^1_E$ and y' either y or $\bar{v}^2_E$. Notice that the size of $\mathcal{B}$ is exponential in the size of γ. Indeed, let k be the number of atoms in γ and let m be an upper bound on the number of states of each 1NFA in γ. All possible variants of a 1NFA obtained by changing the initial state and/or final state are m^2. Hence, the number of possible sets of 1NFAs of at most k elements is $(m^2)^k = 2^{O(m \cdot k)}$.

The automaton $\mathbf{A}^{\gamma}_{Q,\Pi}$ is $(\Sigma, S \cup \{accept\}, S_Q, \delta, \{accept\})$.

– The alphabet Σ is $\mathcal{I} \times \mathcal{R}$. Recall that $\mathcal{I}$ is the set of all IDB atoms with variables among $var(\Pi)$, and $\mathcal{R}$ is the set of instances of rules of Π over $var(\Pi)$.
– The state set S is the set $\mathcal{I} \times \mathcal{B} \times 2^{V_\gamma \times var(\Pi)} \times 2^{V^+_\gamma \times var(\Pi)}$. The second component represents the collection of automata accepting sequences of atoms that have to be mapped to atoms in the tree τ accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$, and the third and fourth component contain the set of partial mappings respectively from V_γ and V^+_γ to $var(\Pi)$.
– The start-state set S_Q consists of all tuples $(Q(\mathbf{s}), \gamma, M_{\gamma,\mathbf{s}}, \emptyset)$, where the variables of $\mathbf{s}$ are in $var(\Pi)$ and $M_{\gamma,\mathbf{s}}$ is a mapping of the distinguished variables of γ into the variables of $\mathbf{s}$.

The transition function δ of $\mathbf{A}^{\gamma}_{Q,\Pi}$ is constructed as follows. Let ϱ be the body of a rule instance in $\mathcal{R}$

$$R(\mathbf{t}) \leftarrow R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$$

1. There is an "atom mapping" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M'_+) \rangle \in \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

   if there is an EDB atom $e(a,b)$ among $R_1(\mathbf{t}^1), \ldots, R_m(\mathbf{t}^m)$ and if β' coincides with β, except that one element $E^f_s(x,y)$ in β is replaced in β' by $E^f_{s'}(x',y)$, and one of the following holds:
   – $s' \in \delta_E(s,e)$ and
      • if $x \in V_\gamma$ (i.e., x is a variable of γ), M maps x to a, and M_+ does not map $\bar{v}^1_E$, then $x' = \bar{v}^1_E$ and $M'_+ = M_+ \cup \{(\bar{v}^1_E, b)\}$;

  - if $x = \bar{v}_E^1 \in V_\gamma^+$ (i.e., x is the first special variable for the 1NFA E) and $(\bar{v}_E^1, a) \in M_+$, then $x' = x = \bar{v}_E^1$, and $M'_+ = M_+ \setminus \{(\bar{v}_E^1, a)\} \cup \{(\bar{v}_E^1, b)\}$;

– $s' \in \delta_E(I, e^-)$ and
  - if $x \in V_\gamma$ (i.e., x is a variable of γ), M maps x to b, and M_+ does not map $\bar{v}_E^1$, then $x' = \bar{v}_E^1$ and $M'_+ = M_+ \cup \{(\bar{v}_E^1, a)\}$;
  - if $x = \bar{v}_E^1 \in V_\gamma^+$ (i.e., x is the first special variable for the 1NFA E) and $(\bar{v}_E^1, b) \in M_+$, then $x' = x = \bar{v}_E^1$, and $M'_+ = M_+ \setminus \{(\bar{v}_E^1, b)\} \cup \{(\bar{v}_E^1, a)\}$.

Intuitively, an "atom mapping" transition maps the next atom recognized by some 1NFA in β to some EDB atom in ρ, and modifies M_+ accordingly. Note that the variable x (either a variable of V_γ or the special variable $\bar{v}_E^1$) must already be mapped (respectively by M or M_+) to some variable in the current node of τ.

2. There is a "splitting" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M'_+)\rangle \ \wedge\ \langle 0, (R(\mathbf{t}), \beta'', M, M''_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if the following hold:

– M'_+ and M''_+ coincide with M_+, except for the changes described in the following point;

– β can be partitioned into β_1, β_2, and β_3; moreover $\beta' = \beta_1 \cup \beta'_3$ and $\beta'' = \beta_2 \cup \beta''_3$, where β'_3 and β''_3 are sets of elements that consist of one element for each element $E_s^f(x, y)$ in β_3, obtained as follows: for some state s' of E and some variable $a \in \mathit{var}(\Pi)$ appearing in $R(\mathbf{t}) \leftarrow \varrho$, one of the following holds:
  - β'_3 contains the element $E_s^{s'}(x, \bar{v}_E^2)$, β''_3 contains the element $E_{s'}^f(\bar{v}_E^1, y)$, M'_+ (re-)maps $\bar{v}_E^2$ to a, and M''_+ (re-)maps $\bar{v}_E^1$ to a;
  - β'_3 contains the element $E_{s'}^f(\bar{v}_E^1, y)$, β''_3 contains the element $E_s^{s'}(x, \bar{v}_E^2)$, M'_+ (re-)maps $\bar{v}_E^1$ to a, and M''_+ (re-)maps $\bar{v}_E^2$ to a;

– β' and β'' can share a variable in V_γ only if this variable is in the domain of M. (Notice that two occurrences of a special variable in V_γ^+ shared by β' and β'' are not related to each other.)

A "splitting" transition partitions the atoms in β into two parts. The goal is to enable the two parts to be manipulated separately. For example, one part may correspond to those atoms that are intended to be "moved" together to an adjacent node in a future transition, while the other part may correspond to those atoms that are meant to stay together in the current node for further processing, e.g., by further splitting or by mapping to EDB atoms. During splitting, some atoms in β may be actually split into two subatoms. The mappings M and M_+ have to "bind" together variables that are in common to the two conjuncts of the transition.

3. There is a "moving" transition

$$\langle j, (R_{i_j}(\mathbf{t}^{i_j}), \beta, M, M_+)\rangle \ \in \ \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

with $j \in \{-1, 1, \ldots, \ell\}$, where ℓ is the number of IDB atoms in ϱ, the atom $R_{i_j}(\mathbf{t}^{i_j})$, for $j \in \{1, \ldots, \ell\}$, is the j-th IDB atom, $R_{i_{-1}}$ stands for R, and $\mathbf{t}^{i_{-1}}$ stands for $\mathbf{t}$, if for all variables that occur in β and that are in the domain of either M or M_+, their image is in $\mathbf{t}^{i_j}$.
A "moving" transition moves to an adjacent node, and is intended to be applied whenever no next atom can be mapped and no further splitting is possible. Moving is possible only if variables that are both in atoms still to be mapped (and thus in β) and have already been mapped (and thus are in the domain of either M or M_+) can be propagated through the head of the rule where the automaton moves.

4. There is an "equality checking" transition

$$\langle 0, (R(\mathbf{t}), \beta', M, M_+)\rangle \;\in\; \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

if the following hold:
   - β can be partitioned into β_0 and β';
   - for all atoms $E^f_s(x, y) \in \beta_0$ we have that
     - $s = f$,
     - (x, a) and (y, a) are in $M \cup M_+$, for some variable a in ϱ or $\mathbf{t}$, i.e., both x and y are in the domain of M or of M_+ and they are mapped to the same variable a;

   An "equality checking" transition gets rid of those elements in β all of whose atoms have already been mapped to atoms in τ. While doing so, it checks that M and M_+ are compatible with the equalities induced by such atoms.
5. There is a "mapping extending" transition

$$\langle 0, (R(\mathbf{t}), \beta, M', M_+)\rangle \;\in\; \delta((R(\mathbf{t}), \beta, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

   if M' is a partial mapping that extends M.
   A "mapping extending" transition adds some variables to the mapping M. This may be necessary to be able to apply some other transition that requires certain variables to appear in M.
6. There is a "final" transition

$$\langle 0, \mathit{accept}\rangle \;\in\; \delta((R(\mathbf{t}), \emptyset, M, M_+), (R(\mathbf{t}) \leftarrow \varrho))$$

   A "final" transition moves to the accepting state whenever there are no further atoms in β that have to be processed.

It is easy to see that the number of states and transitions in $\mathbf{A}^{\gamma}_{Q,\Pi}$ is exponential in the size of Π and γ. The following two basic lemmas establish the correctness of the above construction.

Lemma 2. *Let τ be a proof tree in $p\_trees(Q, \Pi)$. If there is an expansion φ of γ and a strong containment mapping h from φ to τ, then τ is accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$.*

Lemma 3. *Let τ be a proof tree in $p\_trees(Q, \Pi)$. If τ is accepted by $\mathbf{A}^{\gamma}_{Q,\Pi}$, then there is an expansion φ of γ and a strong containment mapping from φ to τ.*

Theorem 4. *Let Π be a Datalog program with binary EDB predicates in Δ and with goal predicate Q, and let $\Gamma = \cup_i \gamma_i$ be a finite union of C2RPQs γ_i over $\Delta^{\pm}$. Then Π is contained in Γ if and only if*

$$\mathcal{T}(\mathbf{A}_{Q,\Pi}^{p\_trees}) \subseteq \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$$

Proof. By Lemma 1, Π is contained in Γ if and only if for every proof tree $\tau \in p\_trees(Q, \Pi)$ there is a γ_i and an expansion φ of γ_i such that there is a strong containment mapping from φ to τ. By Theorem 3 and Lemmas 2 and 3, the latter conditions is equivalent to $\mathcal{T}(\mathbf{A}_{Q,\Pi}^{p\_trees}) \subseteq \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$.

This allows us to establish the main result of the paper.

Theorem 5. *Containment of a recursive Datalog program in a UC2RPQ is in 2EXPTIME.*

Proof. By Proposition 2, we can construct a 2ATA $\mathbf{A}_{Q,\Pi}^{\Gamma}$, whose size is exponential in the size of Π and Γ, such that $\mathcal{T}(\mathbf{A}_{Q,\Pi}^{\Gamma}) = \bigcup_i \mathcal{T}(\mathbf{A}_{Q,\Pi}^{\gamma_i})$. By Proposition 1, we can construct a 1NTA $\mathbf{A}_{Q,\Pi}^{\neg\Gamma}$, whose size is doubly exponential in the size of Π and Γ, such that a Σ-labeled tree is accepted by $\mathbf{A}_{Q,\Pi}^{\neg\Gamma}$ if and only if it is not accepted by $\mathbf{A}_{Q,\Pi}^{\Gamma}$. By Proposition 3, we can construct a 1NTA $\mathbf{A}_{cont}$, whose size is still doubly exponential in the size of Π and Γ, such that $\mathbf{A}_{cont}$ accepts a Σ-labeled tree if and only if it is accepted by $\mathbf{A}_{Q,\Pi}^{p\_trees}$ but not accepted by any of the $\mathbf{A}_{Q,\Pi}^{\gamma_i}$. By Theorem 4, $\mathbf{A}_{cont}$ is nonempty if and only if Π is not contained in Γ. By Proposition 4, nonemptiness of $\mathbf{A}_{cont}$ can be checked in time polynomial in its size, and hence doubly exponential in the size of Π and Γ. The claim follows.

6 Conclusions

We have presented an upper-bound result for containment of Datalog queries in unions of conjunctive regular path queries with inverse (UC2RPQ). This is the most general known decidability result for containment of recursive queries, apart from the result in [23] for monadic Datalog. The class UC2RPQ has several features that are typical of modern query languages for knowledge and data bases. In particular, it is the largest fragment of query languages for XML data [41] for which containment is known to be decidable [34].

The 2EXPTIME upper-bound result shows that adding transitive closure to conjunctive queries does not increase the complexity of query containment with respect to Datalog queries, as it matches the bound obtained in [25] for containment of Datalog queries in union of conjunctive queries. For containment in union of conjunctive queries, the 2EXPTIME bound is shown in [25] to be tight. It is an open question whether our bound here is also tight. The lower bound in [25] is shown using relation symbols of arity up to 8. If that arity can be reduced to 2, then it would follow that our bound here is tight. We conjecture this to be the case. Currently, we have an EXPSPACE lower bound that directly follows from EXPSPACE-completeness of containment of UC2RPQs [34] (which is a special case of containment of Datalog in UC2RPQs). Observe that containment in the converse direction, as well as equivalence, is undecidable already for RPQs. Indeed, universality of context free grammars can be reduced to containment of RPQs in Datalog, by following the line of the undecidability proof of containment between Datalog queries in [22].

Query containment is typically the first step in addressing various problems of query processing, such as view-based query processing. We predict that the decidability result for containment obtained in this paper would prove useful for a broad range of query processing applications.

Acknowledgements. The first and second author were supported in part by MIUR project D2I (Integration, Warehousing and Mining of Heterogeneous Data Sources), by EU Project INFOMIX (Boosting Information Integration) IST-2001-33570, and by EU Project SEWASIE (Semantic Webs and AgentS in Integrated Economies) IST-2001-34825. The third author was supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264.

References

1. Buchheit, M., Jeusfeld, M.A., Nutt, W., Staudt, M.: Subsumption between queries to object-oriented databases. Information Systems **19** (1994) 33–54 Special issue on Extending Database Technology, EDBT'94.
2. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics. In Brewka, G., ed.: Principles of Knowledge Representation. Studies in Logic, Language and Information. CSLI Publications (1996) 193–238
3. Gupta, A., Ullman, J.D.: Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In: Workshop on Deductive Databases (In conjunction with JICSLP), Washington D.C. (USA) (1992) 195
4. Levy, A.Y., Sagiv, Y.: Semantic query optimization in Datalog programs. In: Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95). (1995) 163–173
5. Chaudhuri, S., Krishnamurthy, S., Potarnianos, S., Shim, K.: Optimizing queries with materialized views. In: Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95), Taipei (Taiwan) (1995)
6. Adali, S., Candan, K.S., Papakonstantinou, Y., Subrahmanian, V.S.: Query caching and optimization in distributed mediator systems. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data. (1996) 137–148
7. Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization technique for unstructured data. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data. (1996) 505–516
8. Motro, A.: Panorama: A database system that annotates its answers to queries with their properties. J. of Intelligent Information Systems **7** (1996)
9. Levy, A.Y., Rousset, M.C.: Verification of knowledge bases: a unifying logical view. In: Proc. of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems, Leuven, Belgium (1997)

10. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description logic framework for information integration. In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98). (1998) 2–13
11. Fernandez, M.F., Florescu, D., Levy, A., Suciu, D.: Verifying integrity constraints on web-sites. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99). (1999) 614–619
12. Friedman, M., Levy, A., Millstein, T.: Navigational plans for data integration. In: Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99), AAAI Press/The MIT Press (1999) 67–73
13. Milo, T., Suciu, D.: Index structures for path expressions. In: Proc. of the 7th Int. Conf. on Database Theory (ICDT'99). Volume 1540 of Lecture Notes in Computer Science., Springer (1999) 277–295
14. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77). (1977) 77–90
15. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading, Massachussetts (1995)
16. Chandra, A.K., Harel, D.: Horn clause queries and generalizations. J. of Logic and Computation **2** (1985) 1–15
17. Moschovakis, Y.N.: Elementary Induction on Abstract Structures. North-Holland Publ. Co., Amsterdam (1974)
18. Aho, A.V., Sagiv, Y., Ullman, J.D.: Equivalence among relational expressions. SIAM J. on Computing **8** (1979) 218–246
19. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. of the ACM **27** (1980) 633–655
20. Klug, A.C.: On conjunctive queries containing inequalities. J. of the ACM **35** (1988) 146–160
21. van der Meyden, R.: The Complexity of Querying Indefinite Information. PhD thesis, Rutgers University (1992)
22. Shmueli, O.: Equivalence of Datalog queries is undecidable. J. of Logic Programming **15** (1993) 231–241
23. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs. In: Proc. of the 20th ACM SIGACT Symp. on Theory of Computing (STOC'88). (1988) 477–490
24. Sagiv, Y.: Optimizing Datalog programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos (1988) 659–698
25. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive datalog programs. J. of Computer and System Sciences **54** (1997) 61–78
26. Chaudhuri, S., Vardi, M.Y.: On the complexity of equivalence between recursive and nonrecursive Datalog programs. In: Proc. of the 13th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'94). (1994) 107–116
27. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0 — W3C recommendation. Technical report, World Wide Web Consortium (1998) Available at `http://www.w3.org/TR/1998/REC-xml-19980210`.
28. Calvanese, D., De Giacomo, G., Lenzerini, M.: Representing and reasoning on XML documents: A description logic approach. J. of Logic and Computation **9** (1999) 295–318
29. Buneman, P.: Semistructured data. In: Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97). (1997) 117–121

30. Florescu, D., Levy, A., Mendelzon, A.: Database techniques for the World-Wide Web: A survey. SIGMOD Record **27** (1998) 59–74
31. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: from Relations to Semistructured Data and XML. Morgan Kaufmann, Los Altos (2000)
32. Abiteboul, S., Vianu, V.: Regular path queries with constraints. J. of Computer and System Sciences **58** (1999) 428–452
33. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). (1998) 139–148
34. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000). (2000) 176–185
35. Slutzki, G.: Alternating tree automata. In: Theoretical Computer Science. Volume 41. (1985) 305–318
36. Maier, D., Ullman, J.D., Vardi, M.Y.: On the foundations of the universal relation model. ACM Trans. on Database Systems **9** (1984) 283–308
37. Naughton, J.F.: Data independent recursion in deductive databases. J. of Computer and System Sciences **38** (1989) 259–289
38. Doner, J.E.: Tree acceptors and some of their applications. J. of Computer and System Sciences **4** (1970) 406–451
39. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second order logic. Mathematical Systems Theory **2** (1968) 57–81
40. Costich, O.L.: A Medvedev characterization of sets recognized by generalized finite automata. Mathematical Systems Theory **6** (1972) 263–267
41. Deutsch, A., Fernandez, M.F., Florescu, D., Levy, A., Maier, D., Suciu, D.: Querying XML data. Bull. of the IEEE Computer Society Technical Committee on Data Engineering **22** (1999) 10–18