Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 10 pages

Graph Grammars for Querying Graph-like Data

S. Flesca, F. Furfaro, S. Greco

DEIS, Università della Calabria,
87036 Rende (CS), Italy
{flesca,furfaro,greco}@si.deis.unical.it

Abstract

Recently research has deeply investigated the problem of querying semi-structured data and data which can be represented by means of graphs (e.g. object-oriented data, XML data, etc.). Typically queries on graph-like data, called path queries, are expressed by means of regular expressions denoting paths in the graph. The result of a path query is the set of nodes reachable by means of a path expressed by a specified regular expression. In this paper we investigate the problem of extracting a subgraph satisfying a given property from a given graph representing some information. We propose a new form of queries, called graph queries, whose answers are (marked) graphs having a particular structure, extracted from the source graph. We show that a simple form of graph grammars can be profitably used to define graph queries. The result of a graph query, using a grammar G over a database D, is a marked subgraph of D 'matching' a graph derived from G. We consider different types of graph grammars which can be used to query graph-like data and consider their expressiveness and complexity.

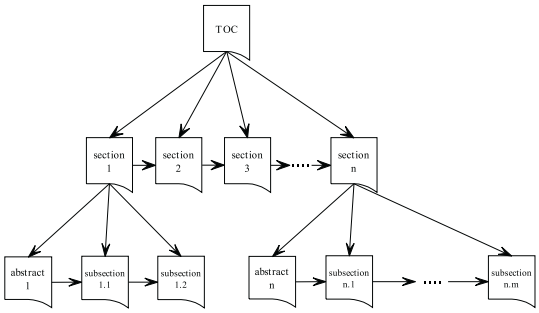
1 Introduction

The widespread use of the web has renewed the interest in problems that had been already investigated with different aims in other contexts, and particularly in the problem of querying graph-like data. Indeed, graphs can be used as an abstract model to represent a wide variety of data, such as semistructured documents, object-oriented data, XML data, etc. [1,2,6,18]. Many queries that can be easily expressed on a relational database are not so "natural" on graph-like data. Accessing this type of data usually requires to navigate the graph searching for the desired information. Recently, several languages and prototypes have been proposed for searching graph-like data such as the Web [1,4,5,8,13,19,20,22]. All these languages permit us to express (declarative) navigational queries, called path queries, whose answer is the set of nodes of the graph which are reachable from a certain node by means of a path speci-

fied by a regular expression [4,7,9,11,23,16]. However, this kind of navigational queries is not completely satisfactory since in many cases we would like to express queries verifying whether the graph has a given structure (e.g. a tree or a chain) or to extract from the source graph a subgraph which satisfies some property.

In this paper we investigate the problem of extracting a subgraph (a consistent subset of nodes and edges) satisfying some property from a given data graph. We propose a new form of queries, called *graph queries*, whose answers are (marked) subgraphs having a particular structure. A graph query is based on a graph grammar [12], that is used to define the structural properties of the subgraph that has to be extracted. A graph grammar is a graph rewriting system consisting of a set of rewriting rules (or *productions*). Like a production of a standard grammar defines how to substitute a non terminal symbol (or a group of symbols) with a string, a production of a graph grammar defines how to replace a node (or an edge) in a graph with a sub-graph. A graph grammar defines a class of graphs which have common structural properties (e.g. the class of complete graphs, the class of trees, etc.): such classes are named *graph languages*. Discussing whether a graph belongs to a certain graph language is equivalent to discussing whether the structure of such a graph satisfies the structural property of that language. For example, we can state that a certain graph is a tree by simply defining a graph grammar generating trees and then demonstrating that the graph belongs to the language generated by that grammar.

Our paradigm permits to define queries that search in a given graph for a subgraph belonging to the language defined by a given graph grammar. In this way it is novel: while other query languages intend to find some nodes in a graph such that each of these nodes has a certain property (e.g. the set of the nodes which are the answer of a path query), our model searches for entire sub-structures. Let's consider the following example. Assume that we want to extract all the available online "books" about *swing* classes from a web site containing documentation about *java*. In particular we are interested in extracting web pages having a hierarchical structure like the one shown in the following figure:



Path queries fail to identify an online book structured in this way, whereas it is easy to describe a structure like this using a graph grammar.

In the following sections we define our graph query language, by introducing a restricted form of node replacement context-free graph grammars [12], called *parsing graph grammars* and specifying how these grammars identify subgraphs of the original data graph. We point out that our paradigm can also be used to create new graphs. However, in this paper we only consider the extraction of subgraphs.

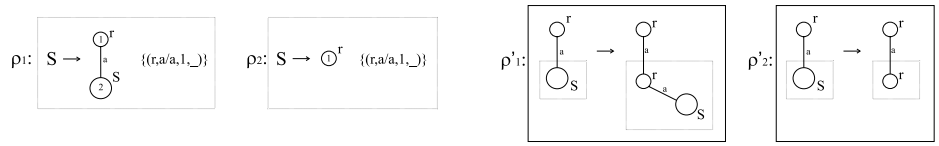
2 NR graph grammars

Node Replacement (NR) graph grammars generate labeled directed graphs. A production of a NR graph grammar is of the form $X \rightarrow (D, C)$ where X is a nonterminal node label, D is a graph and C is the set of connection instructions. A rewriting step of a graph H according to such a production consists of removing a node u labeled X from H, adding D to H and adding edges between D and H as specified by the connection instructions in C. The pair (D, C) can be viewed as a new type of object, and the rewriting step can be viewed as the substitution of the node u with (D, C) in the graph H. Intuitively, these objects are quite natural: they are graphs ready to be embedded in an environment. Their formal definition is as follows.

Let Γ be an alphabet of node labels and Σ an alphabet of edge labels. A *graph with embedding* is a pair $K = (D, C)$ where D is a graph over Γ and Σ and $C \subseteq \Gamma \times \Sigma \times \Sigma \times N \times \{in, out\}$ is the connection relation of K. Each element $(\gamma, \sigma_1, \sigma_2, v, d) \in C$ is a connection instruction of K and is generally written as $(\gamma, \sigma_1/\sigma_2, v, d)$. The components of a graph with embedding K will be denoted as N_K, E_K, λ_K and C_K.

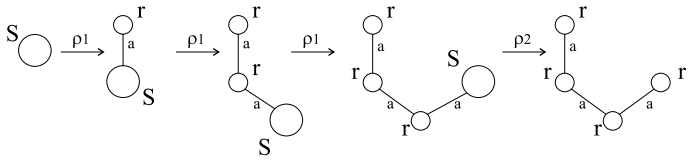
Intuitively, for a graph with embedding K, the meaning of a connection instruction $(\gamma, \sigma_1/\sigma_2, v, out)$ is as follows: if there was a σ_1-labeled edge from a node u which has been substituted by K to a γ-labeled node w, then the embedding mechanism defines a σ_2-labeled edge from v to w. Similarly, the meaning of a connection instruction $(\gamma, \sigma_1/\sigma_2, v, in)$ is as follows: if there was a σ_1-labeled edge from a γ-labeled node w to a node u which has been substituted by K, then the embedding mechanism defines a σ_2-labeled edge from w to v. The feature which replaces edge labels is called *dynamic edge labeling*. Let H be a graph over Γ and Σ, K be a graph with embedding over the same alphabets, and let $v \in N_H$. The substitution of K for v in H is denoted by $H[v/K]$. In the following connection rules of the form $(\gamma, \sigma/\sigma, v, a)$ (i.e. rule which do not re-label edges) are simply written as (γ, σ, v, a).

Example 2.1 *The grammar* G *defined by the following productions* ρ_1 *and* ρ_2 *(or, equivalently, by the productions* ρ'_1 *and* ρ'_2*) describes a language containing chains.:*



ρ'_1 *and* ρ'_2 *have the same meaning, respectively, of* ρ_1 *and* ρ_2*, but in* ρ'_1 *and* ρ'_2 *the connection rules are expressed graphically.*

The following figure illustrates a chain derivation by means of G *productions:*



Definition 2.2 A *node replacement* (NR) grammar is a tuple $G = (\Sigma, \Delta, P, S)$ where Σ is the alphabet of labels, $\Delta \subseteq \Sigma$ is the alphabet of terminal labels, P is the finite set of productions, and $S \in \Sigma - \Delta$ is the initial nonterminal symbol. A production is of the form $X \rightarrow (D, C)$ where $X \in \Sigma - \Delta$ and (D, C) is a graph with embedding. □

The graph appearing in the right side of a production can be empty and a production of the form $X \rightarrow (\emptyset, \emptyset)$ will be simply denoted as $X \rightarrow \epsilon$. Let $G = (\Sigma, \Delta, P, S)$ be an NR grammar. Let H and H' be two graphs, let $v \in N_H$ and let $p : X \rightarrow (D, C)$ be a production of G. Then, we say that H' is directly derived from H (and write $H \Rightarrow_{v,p} H'$, or just $H \Rightarrow H'$), if $\lambda_H(v) = X$ and $H' = H[v/D]$. Moreover, we say that H' is derived from H if there is a finite sequence $H \Rightarrow H_1 \Rightarrow \cdots \Rightarrow H'$.

Thus, a graph grammar defines a class of graphs which have common structural properties. The set of graphs generated by PG is called *graph language* and denoted $\mathcal{L}(PG)$.

3 Querying Data Graphs

We start by defining a simple graph model on an alphabet with two different types of symbols: constant and variables. A variable can take any value and, therefore, it can be associated to any constant. In the following, constants are represented by strings starting with digits or lowercase letters (e.g. $b1$), variable names are denoted by strings preceded by a dollar (e.g. $\$b1$) and non terminal symbols are denoted by strings starting with uppercase letters (e.g. X).

Definition 3.1 *Given an alphabet Σ, a graph over Σ is a tuple $G = (N, E, \lambda)$ where N is a set of nodes, $E \subseteq \{(u, \sigma, v) | u, v \in N, \sigma \in \Sigma\}$ is a set of labeled edges and $\lambda : N \rightarrow \Sigma$ is a node labeling function. We way that G is a* data graph *if Σ contains only constants otherwise it is called* query graph. □

The alphabet used by (general) graphs may also contain, other than terminal symbols (variables and constants), non terminal symbols. Thus, data graphs contain only constants and they are used to represent the input database, query graphs are used to denote graphs which can be 'mapped' on data graphs and general graphs are used in the intermediate states of the derivation process. Graph grammars can be used to denote sub-structures from a given data graph.

Given a graph α, we shall denote with $Terminal(\alpha)$ the sub-graph derived from α by deleting nodes marked with non terminal symbols and arcs connected to deleted nodes. Observe that if the graph α is connected and all terminal symbols does not have out-going arcs, $Terminal(\alpha)$ is also connected.

Example 3.2 *The graph grammar G consisting of the productions of Fig. 1 defines a language consisting of trees.*

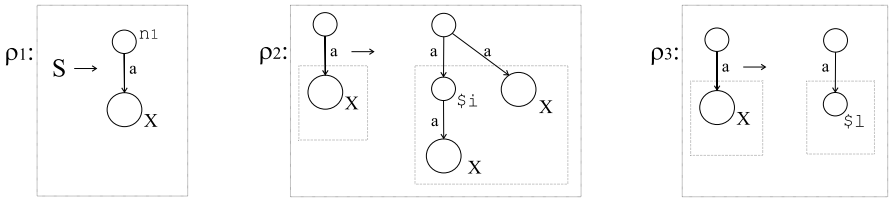


Fig. 1. Graph grammar defining trees

Fig. 2 illustrates a tree derivation by means of G *productions. Note that the*

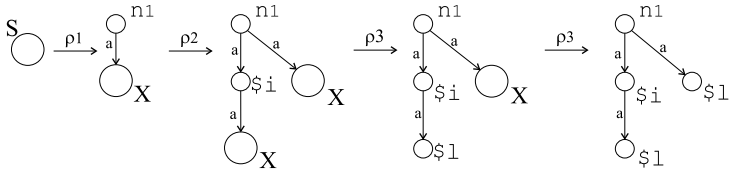


Fig. 2. Graph derivation

root nodes of the trees in the language defined by this grammar have a specific data label (n_1), the internal nodes have label $\$i$ and the leaf nodes have label $\$l$. □

Since in this context we are not interested in generating new graphs, but only in identifying sub-graphs of a given data graph, we shall not consider the whole language generated by a grammar, but only a subset containing graphs which identify some portion of the input data graph. To this purpose we define a mapping from query graphs to sub-graphs of a given data graph.

Definition 3.3 *Let $\alpha = (N, E, \lambda)$ be a query graph over Σ and $D = (N_D, E_D, \lambda_D)$ a data graph over Γ. A* mapping *φ from α to D is a total function mapping, respectively, nodes in N to nodes in N_D and edges in E to edges in E_D such that i) for each node $n \in N$ either $\lambda(n) = \lambda(\varphi(n))$ or $\lambda(n)$ is a variable label, ii) for each arc $(u, \sigma, v) \in E$ there is an arc $(\varphi(u), \gamma, \varphi(v)) \in E_D$ such that either $\sigma = \gamma$ or σ is a variable, and iii) there are no two nodes u and v such that $\lambda(u) = \lambda(v)$ and $\varphi(u) = \varphi(v)$ (i.e. two nodes with the same label cannot by associated to the same node).* □

A data graph D is recognized by a graph grammar PG if there exists a derivation from S to a query graph α ($S \Rightarrow^* \alpha$) and a mapping from α to D. The set of data graphs recognized by PG is denoted $\mathcal{DL}(PG)$. The set of subgraphs of a given data graph D recognized by PG is denoted $\mathcal{DL}(PG, D)$.

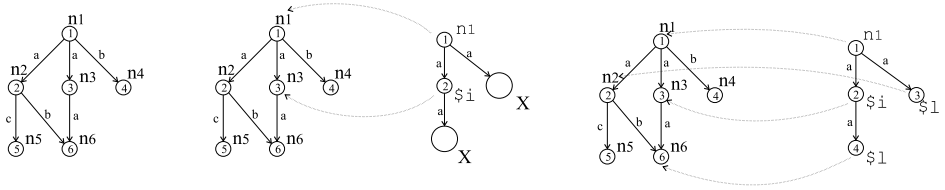
Definition 3.4 *Let D be a data graph. A* mapping pair *on D is a pair (α, φ) where α is a general graph and φ is a data mapping from $Terminal(\alpha)$ to D.* □

Observe that $Terminal(\alpha)$ is a query graph, i.e. a graph whose node labels can be either constant or variables. Like embedded graphs, mapping pairs can be seen as a new type of object consisting of query graphs (derived from graph grammars) mapped over a given data graph. The derivation of query graphs from parsing grammars can be extended to mapping pairs. Let D be a data graph, PG a graph grammar and (α, φ) a mapping pair over D. We say that a mapping pair (β, ψ) is directly derived from (α, φ) through a production ρ of PG (and write $(\alpha, \varphi) \Rightarrow^\rho (\beta, \psi)$) if and only if $\alpha \Rightarrow^\rho \beta$ and ψ extends φ.[1] Moreover, we say that a mapping pair (α_n, φ_n) is derived from a mapping pair (α_0, φ_0) over a data graph D if $(\alpha_0, \varphi_0) \Rightarrow^{\rho_1} (\alpha_1, \varphi_1) \Rightarrow^{\rho_2} \cdots \Rightarrow^{\rho_n} (\alpha_n, \varphi_n)$. Given a graph grammar PG and a data graph D, $\Phi(PG, D)$ defines the set of mapping pairs derived from $(S, \emptyset)$ where $\emptyset$ denotes an empty mapping.

A mapping pair applied to a data graph D allows us to identify a subgraph of D having the property defined by the grammar. Each node of the extracted subgraph can be associated to more than one node of the query graph, if these nodes have different "role" labels. Different labels are used to distinguish different classes of nodes and arcs (e.g. in a tree internal nodes and leaf nodes may have different labels).

[1] $(\varphi \subseteq \psi)$.

Example 3.5 *Consider the parsing grammar of Example 2, the derivation shown in Example 2 and the data graph shown in the left side of the following figure:*



The query graphs produced respectively at the third and at the last step of the derivation can be mapped on D *as shown in the centre and in the right side of Fig 3.5.*
Note that the production defining the axiom (start symbol) of the graph grammar) contains an arc whose start node is marked with the constant label $n1$*. This means that all derived query graphs are trees whose root node is marked with* $n1$*. Therefore, every tree generated by such grammar can be mapped only to a tree whose root node has label* $n1$*. In the above mapping* $\lambda(1) = \lambda(\varphi(1)) = n1$ *whereas all other nodes in the query graph have associated a variable. Although not represented in the figure, the arcs in the query graph are mapped to arcs in the data graph; for instance the arc* $e = (1, 2, a)$ *and the arc* $\varphi(e)$ *have the same label* a *(i.e.* $\varphi(e) = (\varphi(1), \varphi(2), a)$*).* □

Parsing grammars

We now introduce a new type of graph grammars, called *parsing grammars*, which are specialized to extract information from data graphs. Parsing grammars have the following characteristics: 1) the set of production rules is linearly ordered (in order to drive the derivation process and reduce the nondeterminism); 2) a rule can be applied only if a certain condition on the extracted data is satisfied.

Definition 3.6 *A* Parsing (Graph) Grammar *is a tuple* $PG = (\Gamma, \Sigma, P, S)$*, where:* Σ *is an alphabet of terminal symbols,* Γ *is an alphabet of node non terminal symbols (*$\Gamma \cap \Sigma = \emptyset$*),* $S \in \Gamma$ *is the axiom.* P *is a linearly ordered set of productions of the form* $X \rightarrow (\alpha, C)$*, where*

(i) $X \in \Gamma$ *is a non-terminal symbol,*
(ii) α *is a (general) graph in* $\mathcal{G}(\Gamma \cup \Sigma)$*,*
(iii) C *is a set of connection rules, i.e. a set of tuples* (γ, σ, v, d) *where* $d \in \{in, out\}$*,* $\gamma \in \Gamma$*,* $\sigma \in \Sigma$ *and* v *is a node,*
(iv) *for each symbol* $X \in \Gamma$ *there is a production* $X \rightarrow \epsilon$ *in* P*,*
(v) *for each pair of productions* $\rho_i : X \rightarrow (\alpha, C)$ *with* α *not empty and* $\rho_j : X \rightarrow \epsilon$ *is* $\rho_i < \rho_j$ □

Parsing grammars generate query graphs without allowing edge re-labeling. The order on the productions defines an order on derivations of mapping pairs.

Given a data graph D, a parsing grammar PG, and two productions ρ and ν of PG such that $\rho < \nu$, we say that a derivation d_1 of a pair (α_1, φ_1) from a pair (α, φ) precedes a derivation d_2 of a pair (α_2, φ_2) from (α, φ) (written $d_1 \prec d_2$), if 1) $d_1 = (\alpha, \varphi) \Rightarrow^{\rho} (\alpha_i, \varphi_i) \stackrel{*}{\Rightarrow} (\alpha_1, \varphi_1)$, $d_2 = (\alpha, \varphi) \Rightarrow^{\nu} (\alpha_j, \varphi_j) \stackrel{*}{\Rightarrow} (\alpha_2, \varphi_2)$, or 2) there are three derivations d, d_3 and d_4 such that $d_1 = dd_3$ and $d_2 = dd_4$ and $d_3 \prec d_4$.

We can now use the relation $\prec$ to define a partial order on the set of derived mapping pairs $\Phi(PG, D)$. Given two mapping pairs $M_1, M_2 \in \Phi(PG, D)$, we say that $M_1 <_{PG} M_2$ if for each derivation $d_2 = (S, \emptyset) \stackrel{*}{\Rightarrow} M_2$, there exists a derivation $d_1 = (S, \emptyset) \stackrel{*}{\Rightarrow} M_1$ such that $d_1 \prec d_2$. The order introduced on the productions of PG makes $\Phi(PG, D)$ partially ordered. A mapping pair $M \in \Phi(PG, D)$ is said to be *minimal* if there is no mapping pair $M' \in \Phi(PG, D)$ such that $M' <_{PG} M$.

Theorem 3.7 *Let PG be a parsing grammar and D a data graph. A minimal mapping pair in $\Phi(PG, D)$ can be selected nondeterministically in polynomial time.* □

Clearly, any mapping pair in $\Phi(PG, D)$ selected nondeterministically can be also computed in polynomial time.

Graph queries

Parsing grammars have a limited expressive power since they are not able to express NP-complete problems. Their expressiveness can be increased by specifying, separately from the grammar, a *checking property* (that will be expressed as an $FO(COUNT)$ [17] formula) that the extracted graph (i.e. the nodes of the data graph that have been associated to nodes in the generated query graph) must satisfy. $FO(COUNT)$ is an extension of FO containing counting quantifiers [17]. For instance, the meaning of the formula $(\exists i\, x)\ \varphi(x)$ is that there exist at least i distinct elements x such that $\varphi(x)$. In the previous formula the quantifier binds the variable x but leaves i free. For a given data graph D, we assume three relations N_D, E_D and λ containing, respectively, the set of its nodes, the set of its arcs and the association of nodes and arcs to labels. For a given mapping pair (β, φ) on D we consider a unary relation V for each variable v appearing in β which is defined as: $V = \{x | x \in N_D \wedge \exists n \in N_\beta \text{ s.t. } \lambda(n) = v \wedge \varphi(n) = x\}$. The set of all these relations will be denoted as $\mathrm{V}(M)$.

When such a property is defined, the generation process ends successfully only if a terminal graph which satisfies the property has been produced. Otherwise, the generation process must be continued until an "acceptable" graph has been produced or no other graph can be generated.

Definition 3.8 Let D be a data graph D and M a Mapping pair on D. A *checking property* for D and M is a formula of $FO(COUNT)$ over the relations used for storing D and the variables appearing in M. □

The application of a property Π to a data graph D and a mapping pair M on D, denoted by $\Pi(M, D)$, gives a boolean value (*true* if the formula is satisfied and $false$ otherwise).

Example 3.9 *Let D be a data graph D and M a mapping pair on D. The property that M identifies at least k distinct nodes can be expressed by the formula $(\exists_k w)[v_1(w) \vee \cdots \vee v_n(w)]$, where $v_1, \cdots, v_n$ are all the variables that can appear in M. The property that M is a clique is expressed by the formula $(\forall x)(\forall y)\, v(x) \wedge v(y) \wedge (x \neq y) \rightarrow (\exists z)[E_D(x, y, z)]$, whereas the property that the clique has a size greater than or equal to k is expressed by the formula $(\exists_k w)\, v(w) \wedge (\forall x)(\forall y)\, v(x) \wedge v(y) \wedge (x \neq y) \rightarrow (\exists z)[E_D(x, y, z)]$* □

Definition 3.10 An extended graph query Q is a pair (PG, Π) where PG is a graph grammar and Π is a FO(COUNT) formula. The answer to an extended graph query Q over a data graph D is $Q(D) = \{M | M \in \Phi(PG, D) \wedge \Pi(M, D)\}$. □

Theorem 3.11 *Let $Q = (PG, \Pi)$ be an extended graph query and D a data graph. The problem of checking if $Q(D)$ is not empty is NP-complete.* □

Example 3.12 *Consider the parsing grammar G defined by the production on the left side of Fig. 3 (the ε production is omitted), the property $\Pi = (\exists_4 w)(c(w))$ and the data graph D reported on the center of Fig. 3.*

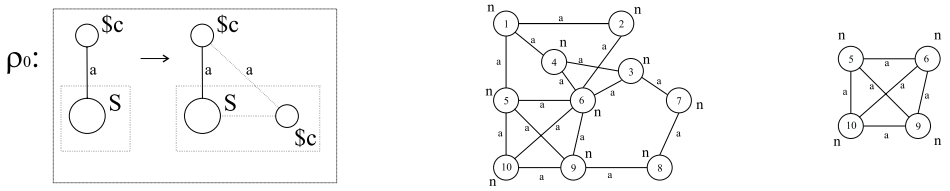


Fig. 3. Extracting a clique

The answer to the query $G(D, \Pi)$ is the clique of size 4 'extracted' from D and reported in the right side of Fig. 3. If the property Π required only 3 nodes in the extracted sub-graph, the query would have identified a set of cliques of size ≥ 3. □

References

[1] Abiteboul, S. Semistructured Data. *Proc. Int. Conf. on Database Theory*, 1997.

[2] Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman, 1999.

[3] Abiteboul S., Hull R., and Vianu V. *Foundations of Databases*. Addison-Wesley, 1994.

[4] Abiteboul S., Vianu V. Regular Path Queries with Constraints. In *Proc. PODS*, pages 122-133, 1997.

[5] Abiteboul S., D. Quass, J. McHugh, J. Widom, J. L. Wiener, The Lorel Query Language for Semistructured Data. in *Journal of Digital Libraries* 1(1), pages 68-88, 1997.

[6] Buneman P. Semistructured Data. *Proc. PODS*. 1997.

[7] Buneman P., Fan W., Weinstein S. Path Constraints in Semistructured and Structured Databases. In *Proc. PODS*, pages 129-138, 1998.

[8] Ceri S., Comai S., Damiani E., Fraternali P., Paraboschi S., Tanca L. XML-GL: A Graphical Language for Querying and Restructuring XML Documents. In *Computer Networks* 31(11-16), pp. 1171-1187, 1999.

[9] Christophides,V., S. Cluet, G. Moerkotte, Evaluating Queries with Generalized Path Expressions, in *Proc. of the ACM SIGMOD Conf.*, pages 413-422, 1996.

[10] Consens M, Mendelzon A., GraphLog: a Visual Formalism for Real Life Recursion. *Proc. Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 404-416, 1990.

[11] Cruz I., Mendelzon A., P. Wood. G^+: Recursive Queries Without Recursion. *Proc. 2nd Int. Conf. on Expert Database Systems*, pages 355-368, 1988.

[12] Engelfriet J. Context-Free Graph Grammars. In *Handbook of Formal Languages, Volume 3: Beyond Words (G. Rozenberg, A. Salomaa, eds.)*, Springer-Verlag, pp. 125-213, 1997.

[13] Fernandez M.F., D. Florescu, J. Kang, A. Y. Levy, D. Suciu, STRUDEL: A Web-site Management System. in *Proc. ACM SIGMOD Conf.*, pages 549-552, 1997.

[14] Flesca, S, and Greco, S. Querying Graph Databases. *Proc. Int. Conf. on Extending Database Technology (EDBT)* 2000, pp. 510-524.

[15] Flesca, S., and Greco, S. Partially Ordered Regular Languages for Graph Queries. *Proc. Int. Colloquium on Automata, Languages and Programming (ICALP)* 1999, pp. 321-330.

[16] Grahne G., Thomo A. An Optimization Technique for Answering Regular Path Queries. In *Proc. of the Third International Workshop on the Web and Databases (WebDB)*, pp. 99-104, 2000.

[17] Immerman, N. *Descriprive Complexity*. Springer-Verlag, 1999.

[18] Kifer, M., W. Kim, Y. Sagiv, Querying Object-Oriented Databases, *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 393-402, 1992.

[19] Konopnicki, D., O. Shmueli, W3QS: A Query System for the World-Wide-Web, in *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 54-65, 1995.

[20] Lakshmanan L., F. Sadri, I. Subramanian, A declarative language for querying and restructuring the web, *Proc. 6th Int. Work. on Research Issues in Data Engineering, RIDE'96*, 1996.

[21] Mecca G., Atzeni P., Masci A., Merialdo P. and Sindoni G. The Araneus Web-Base Management System. In *Proc. of SIGMOD Conference*, pp. 544-546, 1998.

[22] Mendelzon A., G. Mihaila, T. Milo, Querying the World Wide Web, *Journal of Digital Libraries*, 1997.

[23] Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. In *SIAM Journal on Computing*. 24(6), pp. 1235-1258, 1995.

[24] Yannakakis M., Graph-theoretic methods in database theory, In *Proc. Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 230-242, 1990.