Model-Checking Based Data Retrieval

Agostino Dovier[1] and Elisa Quintarelli[2]

[1] Dip. di Matematica e Informatica, Università di Udine.
Via delle Scienze 206, 33100 Udine (IT). dovier@dimi.uniud.it
[2] Dip. di Elettronica e Informazione, Politecnico di Milano.
Piazza Leonardo da Vinci 32, 20133 Milano (IT). quintare@elet.polimi.it

Abstract. In this paper we develop a new method for solving queries on semistructured data. The main idea is to see a database as a Kripke Transition System (a model) and a query as a formula of the temporal logic *CTL*. In this way, the retrieval of data fulfilling a query is reduced to the problem of finding out the states of the model which satisfy the formula (the model-checking problem) that can be done in linear time.
Keywords. Semistructured DBs, Temporal Logic, Model-Checking.

1 Introduction

Most of the information accessible through the Web are typically *semistructured*, i.e. neither raw data nor strictly typed data [1]. It is a common approach to represent semistructured data by using directed labeled graphs [4, 28, 7]. A lot of work has been done to face the problem of accessing in a uniform way this kind of data with graph-based queries. In some approaches queries are really graphs [19, 12, 27] while, in others, queries can be written in extended SQL languages [3, 2, 6]. In both cases, the data retrieval activity requires the development of graph algorithms. In fact, queries (graphical or not) are expected to extract information stored in labeled graphs. In order to do that, it is required to perform a kind of *matching* of the "query" graph with the "database instance" graph. More in detail, we need to find subgraphs of the instance of the database that match (e.g., they are isomorphic or somehow similar to) the query graph. In [13, 4] the concept of similarity used is bisimulation [21]. Even if the problem of establishing whether two graphs are bisimilar or not is polynomial time [20, 25], the task of finding subgraphs isomorphic or bisimilar is NP-complete [16] and hence, not applicable to real-life size problems.

Graphical queries can be easily translated into logic formulae. Techniques for translating graphs in formulae have been exploited in literature [5]. The novel ideas of this work are to associate a *modal* logic formula Ψ to a graphical query, and to interpret database instance graphs as *Kripke Transition Systems (KTS)*. We use a modal logic with the same syntax as the temporal logic *CTL*; the notion of different instants of *time* represents the number of links the user needs to follow to reach the information of interest. This way, finding subgraphs of the database instance graph that match the query can be performed by finding nodes of the *KTS* derived from the database instance graph that satisfy the formula Ψ. This is an instance of the *model-checking* problem, and it is well-known that if the formula Ψ belongs to the class *CTL* of formulae, then the problem is decidable and algorithms running in linear time on both the sizes of the *KTS* and the formula can be employed [11].

We identify a family of graph-based queries that are correctly represented by *CTL* formulae. As immediate consequence, an effective procedure for efficiently querying semistructured databases can be directly implemented on a model checker. We use a "toy" query language called $\mathbb{W}$. It can be considered as a representative of several approaches in which queries are graphical or can easily be seen as graphical (cf., e.g., Lorel [2], G-Log [27], GraphLog [12], and UnQL [19]). We will relate $\mathbb{W}$ to UnQL, GraphLog, and G-Log and show the applicability of the method for implementing (parts of) these languages. We have also effectively tested the approach using the model-checker NuSMV.

2 Transition Systems and *CTL*

In this section we recall the main definitions and results of the model-checking problem for the branching time temporal logic *CTL* [18].

Definition 1. *A* Kripke Transition System *(KTS) over a set Π of* atomic propositions *is a structure $\mathcal{K} = \langle \Sigma, \mathcal{A}ct, \mathcal{R}, I \rangle$, where Σ is a set of states, $\mathcal{A}ct$ is a set of actions, $\mathcal{R} \subseteq \Sigma \times \mathcal{A}ct \times \Sigma$ is a total transition relation, and $I : \Sigma \rightarrow \wp(\Pi)$ is an interpretation (we assume, w.l.o.g., that $\Pi \cap \mathcal{A}ct = \emptyset$).*

Definition 2. *Given the sets Π and $\mathcal{A}ct$ of atomic propositions and actions, CTL formulae are recursively defined as follows:*

1. *each $p \in \Pi$ is a CTL formula;*
2. *if φ_1 and φ_2 are CTL formulae, $a \subseteq \mathcal{A}ct$, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\mathsf{AX}_a(\varphi_1)$, $\mathsf{EX}_a(\varphi_1)$, $\mathsf{AU}_a(\varphi_1, \varphi_2)$, and $\mathsf{EU}_a(\varphi_1, \varphi_2)$ are CTL formulae.*[1]

A and E are the universal and existential path quantifiers, while X (neXt) and U (Until) are the linear-time modalities. Composition of formulae of the form $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, and the modalities F (Finally) and G (Generally) can be defined in terms of the *CTL* formulae: $\mathsf{F}(\varphi) = \mathsf{U}(\mathsf{true}, \varphi)$, $\mathsf{G}(\varphi) = \neg\mathsf{F}(\neg\varphi)$ (cf. [18]).

A *path* (fullpath in [18]) in a KTS $\mathcal{K} = \langle \Sigma, \mathcal{A}ct, \mathcal{R}, I \rangle$ is an infinite sequence $\pi = \langle \pi_0, a_0, \pi_1, a_1, \pi_2, a_2, \ldots \rangle$ of states and actions (π_i denotes the i-th state in the path π) s.t. for all $i \in \mathbb{N}$ it holds that $\pi_i \in \Sigma$ and either $\langle \pi_i, a_i, \pi_{i+1} \rangle \in \mathcal{R}$, with $a_i \in \mathcal{A}ct$, or there are not outgoing transitions from π_i and for all $j \geq i$ it holds that a_j is the special action $\circlearrowleft$ (which is not in $\mathcal{A}ct$) and $\pi_j = \pi_i$.

Definition 3. Satisfaction *of a CTL formula by a state s of a KTS $\mathcal{K} = \langle \Sigma, \mathcal{A}ct, \mathcal{R}, I \rangle$ is defined recursively as follows:*

- *If $p \in \Pi$, then $s \models p$ iff $p \in I(s)$. Moreover, $s \models \mathsf{true}$ and $s \not\models \mathsf{false}$;*
- *$s \models \neg\phi$ iff $s \not\models \phi$;*

[1] When a is of the form $\{m\}$ for a single action m, we simply write $\mathsf{A(E)X(U)_m}$.

- $s \models \phi_1 \wedge \phi_2$ *iff* $s \models \phi_1$ *and* $s \models \phi_2$*;*
- $s \models \mathsf{EX}_a(\phi)$ *iff there is a path* $\pi = \langle s, x, \pi_1, \ldots \rangle$ *s.t.* $x \in a$ *and* $\pi_1 \models \phi$*;*
- $s \models \mathsf{AX}_a(\phi)$ *iff for all paths* $\pi = \langle s, x, \pi_1, \ldots \rangle$*,* $x \in a$ *implies* $\pi_1 \models \phi$*;*
- $s \models \mathsf{EU}_a(\phi_1, \phi_2)$ *iff there is a path* $\pi = \langle \pi_0, \ell_0, \pi_1, \ell_1 \ldots \rangle$*, and* $\exists j \in \mathbb{N}$ *s.t.* $\pi_0 = s$*,* $\pi_j \models \phi_2$*, and* $(\forall i < j)$ $(\pi_i \models \phi_1$ *and* $\ell_i \in a)$*;*
- $s \models \mathsf{AU}_a(\phi_1, \phi_2)$ *iff for all paths* $\pi = \langle \pi_0, \ell_0, \pi_1, \ell_1 \ldots \rangle$ *such that* $\pi_0 = s$*,* $\exists j \in \mathbb{N}$ *s.t.* $\pi_j \models \phi_2$ *and* $(\forall i < j)(\pi_i \models \phi_1$ *and* $\ell_i \in a)$*.*

Definition 4. *The* model-checking problem *can be stated in two instances. The* local *model-checking: given a KTS* $\mathcal{K}$*, a formula* φ*, and a state* s *of* $\mathcal{K}$*, verifying whether* $s \models \varphi$*. The* global *model-checking: given a KTS* $\mathcal{K}$*, and a formula* φ*, finding all states* s *of* $\mathcal{K}$ *s.t.* $s \models \varphi$*.*

If Σ is finite, the global model-checking problem for a *CTL* formula φ can be solved in linear running time on $|\varphi| \cdot (|\Sigma| + |\mathcal{R}|)$ [11].

3 Syntax of the query language $\mathbb{W}$

In this section we describe the syntax of the language $\mathbb{W}$, a very simple graph-based language that we will use to characterize Database queries that have a temporal-logic interpretation.

Definition 5. *A* $\mathbb{W}$-graph *is a directed labeled graph* $\langle N, E, \ell \rangle$*, where* N *is a (finite) set of nodes,* $E \subseteq N \times (\mathcal{C} \times \mathcal{L}) \times N$ *is a set of labeled edges of the form* $\langle m, label, n \rangle$*,* ℓ *is a function* $\ell : N \longrightarrow \mathcal{C} \times (\mathcal{L} \cup \{\perp\})$*.* $\perp$ *means 'undefined', and*

- $\mathcal{C} = \{$ *solid, dashed* $\}$ *denotes how the lines of nodes and edges are drawn.*
- $\mathcal{L}$ *is a set of labels.*

ℓ can be seen as the composition of the two single-valued functions $\ell_{\mathcal{C}}$ and $\ell_{\mathcal{L}}$. With abuse of notation, when the context is clear, we will use ℓ also for edges: if $e = \langle m, \langle c, k \rangle, n \rangle$, then $\ell_{\mathcal{C}}(e) = c$ and $\ell_{\mathcal{L}}(e) = k$. Two nodes may be connected by more than one edge, provided that edge labels be different.

Definition 6. *If* $G = \langle N, E, \ell \rangle$ *is a* $\mathbb{W}$*-graph, then the* size *of* G *is* $|G| = |N| + |E|$*;* $G_s = \langle N_s, E_s, \ell|_{N_s} \rangle$ *is the solid subgraph of* G*, i.e.* $N_s = \{n \in N : \ell_{\mathcal{C}}(n) = solid\}$ *and* $E_s = \{\langle m, \langle solid, \ell \rangle, n \rangle \in E : m, n \in N_s\}$*; given two sets of nodes* $S, T \subseteq N$*,* T *is* accessible *from* S *if for each* $n \in T$ *there is a node* $m \in S$ *such that there is a path in* G *from* m *to* n*.*

Definition 7. *A* $\mathbb{W}$-instance *is a* $\mathbb{W}$*-graph* G *such that* $\ell_{\mathcal{C}}(e) = solid$ *for each edge* e *of* G *and* $\ell_{\mathcal{C}}(n) = solid$ *and* $\ell_{\mathcal{L}}(n) \neq \perp$ *for each node* n *of* G*.*

Definition 8. *A* $\mathbb{W}$-query *is a pointed* $\mathbb{W}$*-graph, namely a pair* $\langle G, \nu \rangle$ *with* ν *a node of* G *(the point). A* $\mathbb{W}$-query $\langle G, \nu \rangle$ *is* accessible *if the set* N *of nodes of* G *is accessible from* $\{\nu\}$*.*

See Fig. 1 for some examples of $\mathbb{W}$-graphs. Dashed nodes and lines are introduced to allow a form of negation. The meaning of the first query is: collect all the teachers aged 37. The second query asks for all the teachers that have declared some age (observe the use of an undefined node). The third query, instead, requires to collect all the teachers that teach some course, but not Databases.

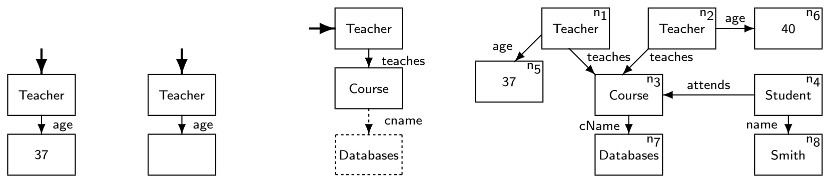


Fig. 1. Three $\mathbb{W}$-queries and a $\mathbb{W}$-instance

4 $\mathbb{W}$-Instances as Kripke Transition System

In this section we show how to build the *KTS* associated with a $\mathbb{W}$-instance.

Definition 9. *Let $G = \langle N, E, \ell \rangle$ be a $\mathbb{W}$-instance; we define the* KTS $\mathcal{K}_G = \langle \Sigma_G, \mathcal{A}ct_G, \mathcal{R}_G, \mathcal{I}_G \rangle$ *over the set of atomic propositions Π_G as follows:*

- *Π_G is the set of all the node labels of G: $\Pi_G = \{p : (\exists n \in N)(p = \ell_{\mathcal{L}}(n)\}$.*
- *The set of states is $\Sigma_G = N$.*
- *The set of actions $\mathcal{A}ct_G$ includes all the edge labels. In order to capture the notion of* before *we also add in $\mathcal{A}ct_G$ actions for the inverse relations and for the negation of all the relations introduced.*[2] *Thus, if $m \xrightarrow{p} n$ belongs to E we add the actions $p, p^{-1}, \bar{p}, \bar{p}^{-1}$. We define two sets: $\mathcal{A}ct^+_G = \{p, p^{-1} : (\exists m \in N)(\exists n \in N)(\langle m, p, n \rangle \in E)\}$ and $\mathcal{A}ct_G = \{q, \bar{q} : q \in \mathcal{A}ct^+_G\}$.*
- *The ternary transition relation $\mathcal{R}_G$ is defined as follows: let $\tilde{E}_G = E \cup \{\langle n, p^{-1}, m \rangle : \langle m, p, n \rangle \in E\}$. Then $\mathcal{R}_G = \tilde{E}_G \cup \{\langle m, \bar{p}, n \rangle : m, n \in N, p \in \mathcal{A}ct^+_G, \langle m, p, n \rangle \notin \tilde{E}\}$. Moreover, we can assume, for each state s with no outgoing edge in E (a leaf in G) to add a self-loop edge labeled by the special action $\circlearrowleft$ that is not in $\mathcal{A}ct_G$.*
- *The interpretation function $\mathcal{I}_G$ can be defined as follows. In each state n the only formulae that hold are the unary atom $\ell_{\mathcal{L}}(n)$ and* true*: $\mathcal{I}_G(n) = \{\mathsf{true}, \ell_{\mathcal{L}}(n)\}$.*[3]

Observe that: $|\mathcal{R}_G| = |\mathcal{A}ct^+_G| \cdot |N|^2 \leq |E| \cdot |N|^2$ since for each pair of nodes, exactly one between q or $\bar{q}$ holds, for $q = p$ or $q = p^{-1}$, $q \in \Pi_G$. For instance, consider the graph $G = \langle \Sigma_G = \{n_1, \ldots, n_8\}, E, \ell \rangle$ of Fig. 1. It holds that:

- $\Pi_G = \{\,\text{Teacher}, \text{Course}, \text{Student}, 37, 40, \text{Databases}, \text{Smith}\}$,
- $\mathcal{I}(n_1) = \{\mathsf{true}, \text{Teacher}\}, \mathcal{I}(n_2) = \{\mathsf{true}, \text{Teacher}\}, \ldots, \mathcal{I}(n_8) = \{\mathsf{true}, \text{Smith}\}$.

5 Temporal Logic semantics of $\mathbb{W}$-queries

In this section we show how to extract *CTL* formulae from $\mathbb{W}$-queries. We associate a formula $\Psi_\nu(G)$ to a query (a $\mathbb{W}$-pointed graph) $\langle G, \nu \rangle$. Such a formula

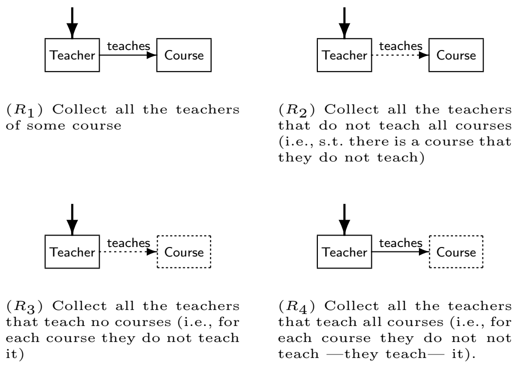
[2] Actually, negated edges are not always needed to be effectively stored—cf. Sect. 6.
[3] The two unary atoms represent the basic local properties of a system state. Other properties can be added, if needed.

allows us the possibility to define a model-checking based methodology for querying a $\mathbb{W}$-instance. We anticipate this definition in order to make the meaning of formula encoding more clear.

Definition 10 (Querying). *Given a $\mathbb{W}$-instance I and a $\mathbb{W}$-query $\langle G, \nu \rangle$, let $\mathcal{K}_I$ be the KTS associated with I and $\Psi_\nu(G)$ the CTL formula associated with $\langle G, \nu \rangle$.* Querying *I with G amounts to solve the global model-checking problem for $\mathcal{K}_I$ and $\Psi_\nu(G)$, namely find all the states s of $\mathcal{K}_I$ such that $s \models \Psi_\nu(G)$.*

5.1 Technique Overview

In order to explain the technique, we start our analysis by considering simple queries in which the pointed graph consists of two nodes (Fig. 2).



(R_1) Collect all the teachers of some course

(R_2) Collect all the teachers that do not teach all courses (i.e., s.t. there is a course that they do not teach)

(R_3) Collect all the teachers that teach no courses (i.e., for each course they do not teach it)

(R_4) Collect all the teachers that teach all courses (i.e., for each course they do not not teach —they teach— it).

Fig. 2. Simple queries

Query R_1 has no dashed part: only positive information is required. Its meaning is to look for nodes labeled Teacher that are connected with nodes labeled Course by an edge labeled teaches. The *CTL* formula must express the statement "In this state Teacher formula is true and there is one next state reachable by an edge labeled teaches, where the Course formula is true", i.e.

$$\mathsf{Teacher} \wedge \mathsf{EX}_{\mathsf{teaches}}(\mathsf{Course})$$

The *CTL* operator *neXt* (used either as EX or AX) captures the notion of following an edge in the graph. Thanks to this operator, we can easily define a path on the graph, nesting formulae that must be satisfied by one (or every) next state.

Query R_2 contains a dashed edge teaches that introduces a negative information. R_2 requires the existence of two nodes, and the *non*-existence of one edge labeled teaches between them. We can express this statement by

$$\mathsf{Teacher} \wedge \mathsf{EX}_{\overline{\mathsf{teaches}}}(\mathsf{Course})$$

The availability of the negation of the predicate symbol $\mathsf{teaches}$, allows us to say that the relation $\mathsf{teaches}$ does not hold between two nodes is the same as requiring that, between the same pair of nodes, the relation $\overline{\mathsf{teaches}}$ holds.

The meaning of query R_3, where there are dashed edges and nodes, is rather different. This formula is true if "there is a node labeled $\mathsf{Teacher}$ s.t., for all the nodes labeled Course, the relation $\mathsf{teaches}$ is not fulfilled". A *CTL* formula that states this property is the following:

$$\mathsf{Teacher} \wedge \mathsf{AX}_{\mathsf{teaches}}(\neg \mathsf{Course})$$

To give a semantics to query R_4, first replace the solid edge labeled $\mathsf{teaches}$ with the dashed edge labeled $\overline{\mathsf{teaches}}$. Then use the same interpretation as for query R_3:

$$\mathsf{Teacher} \wedge \mathsf{AX}_{\overline{\mathsf{teaches}}}(\neg \mathsf{Course})$$

Its meaning is: "it is true if $\mathsf{Teacher}$ is linked by edges labeled $\mathsf{teaches}$ to all the Course nodes of the graph". Note the extremely compact way for expressing universal quantification.

5.2 Admitted queries

We will show how to encode $\mathbb{W}$-queries in *CTL* formulae. The equivalence result with G-log (Sect. 7) and the NP-completeness of the subgraph bisimulation problem ([16]) prevents us to encode all possible queries in a framework that can be solved in polynomial time. We will encode four families of queries $Q = \langle G, \mu \rangle$:

- Q is an acyclic *accessible* query (Sect. 5.4).
- G is an acyclic *solid* graph (Sect. 5.4).
- G is an acyclic graph and after the application of a rewriting procedure, it becomes acyclic and accessible from $\{\mu\}$ (Sect. 5.4).
- G is in one of the forms above with some leaf nodes replaced by simple solid cycles (Sect. 5.5).

5.3 Query Translation

As initial step, we associate a formula φ to each node and edge of a graph G. Then we will use this notion for the definition of the formula.

Definition 11. *Let $G = \langle N, E, \ell \rangle$ be a $\mathbb{W}$-graph. For all nodes $n \in N$ and for all edges $e = \langle n_1, \langle c, p \rangle, n_2 \rangle \in E$, we define:*

$$\varphi(n) = \begin{cases} \ell_{\mathcal{L}}(n) \text{ if } \ell_{\mathcal{L}}(n) \neq \bot \\ \mathsf{true} \quad \text{otherwise} \end{cases} \qquad \varphi(e) = \begin{cases} p & \text{if } \ell_{\mathcal{C}}(e) = \ell_{\mathcal{C}}(n_2) \\ \overline{p} & \text{otherwise} \end{cases}$$

5.4 Acyclic Graphs

Definition 12. *Let $G = \langle N, E, \ell \rangle$ be an acyclic $\mathbb{W}$-graph, and $\nu \in N$. The formula $\Psi_\nu(G)$ is defined recursively as follows (cf. Fig. 3):*

- *let $b_1, \ldots, b_h$ $(h \geq 0)$ be the successors of ν s.t. $\ell_{\mathcal{C}}(b_i) = solid$,*
- *let $c_1, \ldots, c_k$ $(k \geq 0)$ those s.t. $\ell_{\mathcal{C}}(c_i) = dashed$,*

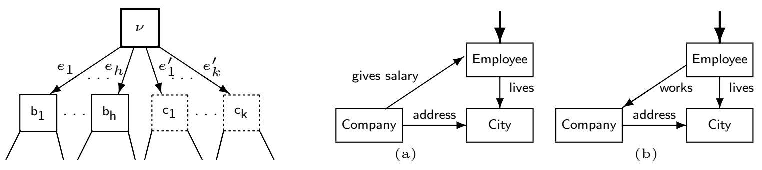


Fig. 3. Graph for computing $\Psi_\nu(G)$ and two equivalent acyclic queries

– *for* $i = 1, \dots, h$ *and* $j = 1, \dots, k$ *let* e_i *be the edge which links* ν *to* b_i *and* e'_j *the one which links* ν *to* c_j. *If* $\ell_{\mathcal{C}}(\nu) = solid$, *then:*

$$\Psi_\nu(G) = \varphi(\nu) \wedge \bigwedge_{i=1\dots h} \mathsf{EX}_{\varphi(e_i)}(\Psi_{b_i}(G)) \wedge \bigwedge_{j=1\dots k} \mathsf{AX}_{\varphi(e'_j)}(\Psi_{c_j}(G))$$

else ($\ell_{\mathcal{C}}(\nu) = dashed$)*:*

$$\Psi_\nu(G) = \neg\varphi(\nu) \vee \bigvee_{i=1\dots h} \mathsf{AX}_{\varphi(e_i)}(\Psi_{b_i}(G)) \vee \bigvee_{j=1\dots k} \mathsf{EX}_{\varphi(e'_j)}(\Psi_{c_j}(G))$$

Given a graph G, let $\bar{G}$ be the graph obtained from G by complementation of the colors of edges and nodes (solid becomes dashed and vice versa). It is immediate to prove, by induction on the depth of the subgraph of G that can be reached from a node ν, that $\Psi_\nu(G) = \neg\Psi_\nu(\bar{G})$. Moreover, by the recursive definition of the formula, and by the acyclicity of G, $\Psi_\nu(G)$ is a *CTL* formula.

Definition 13. *Let* $G = \langle N, E, \ell \rangle$ *be an acyclic* $\mathbb{W}$*-graph,* $\nu \in N$*, and* $Q = \langle G, \nu \rangle$ *be an accessible query. The formula associated to* Q *is* $\Psi_\nu(G)$.

Observe that each node and edge of G is used to build the formula.

Remark 1. The size of the formula $\Psi_\nu(G)$ can grow exponentially w.r.t. $|G|$, since the construction of the formula involves the unfolding of a DAG. However, it is only a representation problem: It is easy to compute the formula avoiding repetitions of subformulae and keeping the memory allocation linear w.r.t. $|G|$.

The condition on the accessibility of all nodes of G for ν can be weakened. Consider, for instance, the two goals of Fig. 3. If works is the inverse relation of gives salary (i.e. gives salary^{-1}), then one expects the same results from queries (a) and (b). Thus, the idea is to swap the direction of some edges, replacing the labeling relation with its inverse.[4]

Algorithm 1 *Let* $G = \langle N, E, \ell \rangle$ *be an acyclic solid* $\mathbb{W}$*-graph and* $\nu \in N$.

1. *Let* $\hat{G} = \langle N, \hat{E} \rangle$ *be the non-directed graph obtained from* G *defining* $\hat{E} = \{\{m, n\} : \langle m, \ell, n \rangle \in E\}$.
2. *Identify each connected component of* $\hat{G}$ *by one of its nodes. Use* ν *for its connected component. Let* $\mu_1, \dots, \mu_h$ *be the other chosen nodes.*

[4] Recall that in the KTS associated to a $\mathbb{W}$-instance, inverse relations for all the relations involved occur explicitly: the framework is tuned to deal also with this case.

3. *Execute a breadth-first visit of $\hat{G}$ starting from ν, $\mu_1, \ldots, \mu_h$.*
4. *Consider the list of nodes $\nu = n_0 < n_1 < \cdots < n_k$ ordered by the above visit.*
5. *Build the $\mathbb{W}$-graph $\boldsymbol{G} = \langle N, \boldsymbol{E}, \ell \rangle$ from G as follows:*

$$\begin{aligned} \boldsymbol{E} = (E \setminus \{ \langle n_a, \langle c, p \rangle, n_b \rangle \in E : b < a \}) \cup \\ \{ \langle n_b, \langle c, p^{-1} \rangle, n_a \rangle : \langle n_a, \langle c, p \rangle, n_b \rangle \in E \wedge b < a \} \end{aligned}$$

The above algorithm always produces an acyclic graph, since the edges follow a strict order of the nodes. All the nodes of each connected component of $\hat{G}$ are accessible from the corresponding selected node $(\nu, \mu_1, \ldots, \mu_h)$ by construction (they have been reached by a visit). Algorithm 1 can be implemented so as to run in time $O(|N| + |E|)$ and, for each node $\nu, \mu_1, \ldots, \mu_h$, we can compute:

$$\Psi_\nu(\boldsymbol{G}), \Psi_{\mu_1}(\boldsymbol{G}), \ldots, \Psi_{\mu_h}(\boldsymbol{G})$$

We recall here the semantics of $\mathsf{EF}_a(\phi)$ (see Sect. 2): $s \models \mathsf{EF}_a(\phi)$ iff there is a path $\langle \pi_0, \ell_0, \pi_1, \ell_1 \cdots \rangle$ s.t. $\pi_0 = s$ and $\exists j \in \mathbb{N}$ s.t. $\pi_j \models \phi$ and $(\forall i < j)\, \ell_i \in a$.

Definition 14. *Let $G = \langle N, E, \ell \rangle$ be an acyclic solid $\mathbb{W}$-graph and $\nu \in N$. Let $Q = \langle G, \nu \rangle$ a $\mathbb{W}$-query. The formula associated with Q is ($\mathcal{A}ct_G$ is as in Def. 9):*

$$\Psi_\nu(\boldsymbol{G}) \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_1}(\boldsymbol{G})) \wedge \cdots \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_h}(\boldsymbol{G}))$$

Observe that Algorithm 1 terminates even if G admits cycles (save self-loops). However, in these cases, the semantics of the query is lost. Cyclic queries require different modal operators (see Sect. 5.5).

Let us study one more family of acyclic queries that can be handled, via reduction to the accessible query case.

Definition 15. *Let $G = \langle N, E, \ell \rangle$ be an acyclic $\mathbb{W}$-graph, let $\nu \in N$, $\ell_{\mathcal{C}}(\nu) =$ solid, and $Q = \langle G, \nu \rangle$ be a $\mathbb{W}$-query.*

1. *Let $G_s = \langle N_s, E_s, \ell|_{N_s} \rangle$ its solid subgraph (cf. Def. 6).*
2. *Apply the Algorithm 1 to G_s. Let $\nu, \mu_1, \ldots, \mu_h$ be the root nodes.*
3. *Swap in G the same edges that have been swapped by Algorithm 1 in G_s obtaining the graph $\boldsymbol{G}$.*
4. *If $\boldsymbol{G}$ is acyclic, then compute the formulae $\Psi_\nu(\boldsymbol{G}), \Psi_{\mu_1}(\boldsymbol{G}), \ldots, \Psi_{\mu_h}(\boldsymbol{G})$*
5. *If all the nodes of G have been visited during the last phase, then the formula associated with Q is: $\Psi_\nu(\boldsymbol{G}) \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_1}(\boldsymbol{G})) \wedge \cdots \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_h}(\boldsymbol{G}))$*

Let us explain why we applicate the algorithm only to G_s. Consider, for example, the query (a) in Fig. 4. According to Def. 15, its formula is:

$$\mathsf{Teacher} \wedge \mathsf{AX}_{\neg\mathsf{teaches}}(\neg\mathsf{Course}) \wedge \mathsf{EF}_{\mathcal{A}\mathsf{ct}_\mathsf{G}}(\mathsf{Student} \wedge \mathsf{AX}_{\neg\mathsf{attends}}(\neg\mathsf{Course})) \quad (1)$$

which requires to find those Teachers who teach all the Courses, if there is somewhere a Student who attends all the Courses.

The Algorithm 1 should replace the edge labeled $\mathsf{attends}$ in the query (a) with an edge labeled $\mathsf{attends}^{-1}$, as depicted in the query (b) of Fig. 4, leading to:

$$\mathsf{Teacher} \wedge \mathsf{AX}_{\neg\mathsf{teaches}}(\neg\mathsf{Course} \vee \mathsf{AX}_{\mathsf{attends}^{-1}}\mathsf{Student}) \quad (2)$$

The two formulae have different models. (1) is closer than (2) to the interpretation of graph-based formulae in other frameworks (e.g. G-Log).

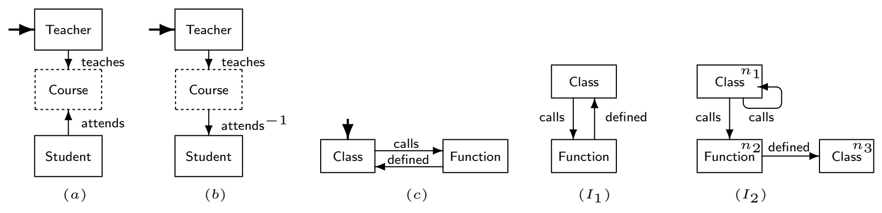


Fig. 4. $\mathbb{W}$-queries and $\mathbb{W}$-instances

5.5 Cyclic queries

In this section we extend the technique assigning a temporal formula to queries admitting cycles. We make use of the Generally operator, used as $\mathsf{EG}_a(\phi)$, whose semantics is (see Sect. 2): $s \models \mathsf{EG}_a(\phi)$ iff there is a path $\pi = \langle \pi_0, \ell_0, \pi_1, \ell_1 \ldots \rangle$, s.t. $\pi_0 = s$ and $\forall j \in \mathbb{N}\ \pi_j \models \phi$ and $\ell_j \in a$.

Consider the query (c) in Fig. 4. It requires to collect *all the classes which call a function defined inside theirselves.* This property could be expressed by:

$$\begin{aligned}\Psi_{(c)} = \mathsf{Class} \wedge \mathsf{EX}_{\mathsf{calls}}(\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{defined}}(\mathsf{Class})) \wedge \\ \mathsf{EG}_{\{\mathsf{calls},\mathsf{defined}\}}((\mathsf{Class} \vee \mathsf{Function}) \wedge \mathsf{Class} \rightarrow \mathsf{X}_{\mathsf{calls}}(\mathsf{Function}) \wedge \\ \mathsf{Function} \rightarrow \mathsf{X}_{\mathsf{defined}}(\mathsf{Class}))\end{aligned}$$

The modal operator X is used without any path quantifier. This is allowed in *CTL** but not in *CTL* [18]. The first part of the formula $\Psi_{(c)}$ is aimed at identifying cycles of length greater than or equal to two, and the Generally operator imposes to retrieve only cyclic paths where nodes labeled Class alternate with nodes labeled Function.

With the *CTL* logic it is only possible to approximate the translation of this kind of cyclic queries:

$$\begin{aligned}\Psi'_{(c)} = \mathsf{Class} \wedge \mathsf{EX}_{\mathsf{calls}}(\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{defined}}(\mathsf{Class})) \wedge \\ \mathsf{EG}_{\{\mathsf{calls},\mathsf{defined}\}}(\mathsf{Class} \rightarrow \mathsf{EX}_{\mathsf{calls}}(\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{defined}}(\mathsf{Class})))\end{aligned}$$

The node Class of instance (I_1) in Fig. 4 satisfies both the formulas $\Psi_{(c)}$ and $\Psi'_{(c)}$, while the node n_1 of instance (I_2) in Fig. 4 satisfies $\Psi'_{(c)}$ but not $\Psi_{(c)}$. Thus, the *CTL* translation gives only an approximation of the *CTL** one.

Since the model-checking problem for *CTL** is PSPACE complete, we accept this loss of precision, and we assign a formula to a (pointed) cycle.

Definition 16. *The formula $\Psi_{\nu_1}(G)$ associated to a red cyclic graph $G = \langle \{\nu_1, \ldots, \nu_n\}, \{\langle \nu_1, \ell_1, \nu_2 \rangle, \langle \nu_2, \ell_2, \nu_3 \rangle, \ldots, \langle \nu_{n-1}, \ell_{n-1}, \nu_n, \rangle, \langle \nu_n, \ell_n, \nu_1 \rangle\}, \ell, \nu_1 \rangle$ is defined as:*

$$\Psi_{\nu_1}(G) = \Psi_{\nu_1}(C) \wedge \mathsf{EG}_{\{\ell_1,\ldots,\ell_n\}}(\varphi(\nu_1) \rightarrow \Psi_{\nu_1}(C))$$

where $\varphi(\nu_1)$ is as in Def. 11, $\Psi_{\nu_1}(C)$ is the formula associated to the DAG $C = \langle \{\nu_1, \ldots, \nu_{n+1}\}, \{\langle \nu_1, \ell_1, \nu_2 \rangle, \langle \nu_2, \ell_2, \nu_3 \rangle, \ldots, \langle \nu_{n-1}, \ell_{n-1}, \nu_n, \rangle, \langle \nu_n, \ell_n, \nu_{n+1} \rangle\}\rangle$, rooted at ν_1, and ν_{n+1} is a "copy" of ν_1, i.e. a new node s.t. $\ell(\nu_{n+1}) = \ell(\nu_1)$.

If a graph G is in one of the forms of the previous sections and, moreover, instead of some leaf nodes it contains cycles of the form above (this test can be easily performed using the graph of the strongly connected components), we can use the formula in Def. 16 as subroutine to compute the global formula.

6 Complexity issues and Implementation of the method

Let us state the main computational results of our approach:

Theorem 2. *Let $\langle G, \nu \rangle$ be a $\mathbb{W}$-query in one of the forms described in Sect. 5.2. Querying a $\mathbb{W}$-instance I with G can be done in linear time on $|\mathcal{K}_I|$ and $|\Psi_\nu(G)|$.*

The proof of the theorem follows from [11]. When computing $\mathcal{K}_I$, a quadratic time and space complexity is introduced as negative relations are computed and stored. We will discuss later on in this section when this extra complexity can be avoided.

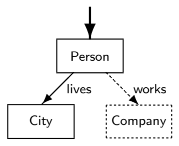
As far as the size of the formula is concerned, as discussed in Remark 1, even if $|\Psi_\nu(G)|$ can grow exponentially with $|G|$, it is natural to represent it using a linear amount of memory. This compact representation is allowed by the model-checker NuSMV.

Corollary 1. *Querying a $\mathbb{W}$-instance I with a $\mathbb{W}$-query G can be done in time linear on $|I|^2$ and $|G|$.*

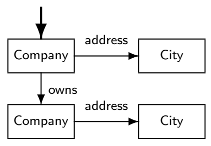
As shown in Sect. 5.5 for cyclic queries, it seems to be impossible to map all the queries in *CTL* formulae. This fact can be formally justified. The semantics of acyclic $\mathbb{W}$-queries without negation can be proved to be equivalent to that of G-Log queries without negation (Sect. 7). If we extended this equivalence result to cyclic queries (even without negation), we would provide a polynomial time implementation for a subset of G-Log in which data retrieval is equivalent to the subgraph bisimulation problem, proved to be NP complete in [16].

We have effectively tested the data retrieval technique presented in the previous sections by using the model checker NuSMV [10]. Due to lack of space, we do not enter here into the details of the implementation (see [17]). We only stress here on the fact that negated edges (relations) are not always needed to be explicitly stored in the model. If the query expresses negation only by means of dashed edges entering dashing nodes, we can avoid to store explicitly negated edges (we translate the formula with an universal quantification on paths, cf. Fig. 2), obtaining linear time complexity instead of quadratic complexity. This fact can be proved using the fact that model-checkers use 'paths' to traverse a KTS and thus to find pairs (or tuples) of nodes connected with a certain edge.

As already said, a formula can be stored as a DAG, without the unnecessary loss of space due to the repetition of subformulae. This allows linear time dependency on the size of the query. Two examples are reported in Fig. 5.



```
SPEC
   l = person & EX(l = lives & EX(l = city))
    & AX(l = works -> ! EX(l = company))
```



```
SPEC
   l = company & EX(l = owns & EX(l = company
    & EX(l = address & EX(l = city))))
    & EX(l = address & EX(l = city))
```

Fig. 5. Translation of Queries in NuSMV

7 Comparison and Applicability of the method

We have studied the applicability of the method to three existing query languages for semistructured data: UnQL, GraphLog, and G-Log. Due to lack of space, we give only a brief overview here. For further details, see [17]. Then we discuss in detail the problems related to the implementation of the *join* operation in our framework.

UnQL is a language for querying data organized as a rooted directed graph with labeled edges [3]. A rooted graph is a pointed graph such that all the nodes are accessible from the root node (the point). UnQL database instances can be immediately and completely mapped to $\mathbb{W}$-instances. Basically, it is sufficient to encode labeled edges into labeled nodes. We replace every labeled edge $m \xrightarrow{label} n$ by the two edges $m \longrightarrow \mu, \mu \longrightarrow n$, where μ is a new node labeled *label*.

In order to encode UnQL queries into modal formulae we have used the modal operator B (Before), whose meaning is the following (we have omitted edge names since all labels have been moved to nodes): $s \models \mathsf{AB}\phi$ iff for all paths $\pi = \langle \dots, x, \_, s, \_, \pi_1, \dots \rangle$, it holds that $x \models \psi$, whereas $s \models \mathsf{EB}\psi$ iff there is a path $\pi = \langle \dots, x, \_, s, \_, \pi_1, \dots \rangle$ such that $x \models \psi$.[5]

The expression (1): **select** t **where** $R1 \Rightarrow \backslash t \leftarrow DB$ computes *the union of all trees t such that DB contains an edge* $R1 \Rightarrow t$ *emanating from the root.* This concept can be expressed by the temporal formula $\varphi_{(1)} = \bot \wedge \mathsf{EB}(R1)$.

The UnQL expression (2): **select** t **where** $\backslash \ell \Rightarrow \backslash t \leftarrow DB$ retrieves *any edge emanating from the root* and is translated by the formula $\varphi_{(2)} = \bot \wedge \mathsf{EB}(\neg\bot \wedge \mathsf{EB}(DB))$.

An UnQL query that looks at arbitrarily depth into the database to find *all edges with a numeric label* is (3): **select** $\{\ell\}$ **where** $\_* \Rightarrow \backslash \ell \Rightarrow \_ \leftarrow DB, isnumber(\ell)$. It can be easily translated into the *CTL* formula $\psi_{(3)} = \neg\bot \wedge number$. Note that *number* is an atomic proposition that holds in numeric states of the original database.

[5] The B operator can be removed (by using X) since inverse relations can be stored.

The UnQL query: **select** $\{Tup \Rightarrow \{A \Rightarrow x, D \Rightarrow z\}\}$
where $R1 \Rightarrow Tup \Rightarrow \{A \Rightarrow \backslash x, C \Rightarrow \backslash y\} \leftarrow DB$,
$R2 \Rightarrow Tup \Rightarrow \{C \Rightarrow \backslash y, D \Rightarrow \backslash z\} \leftarrow DB$

computes *the join of* $R1$ *and* $R2$ *on their common attribute* C *and the project onto* A *and* D. UnQL queries expressing a join condition on a graph do not have a corresponding temporal formula, as we discuss at the end of the section.

Although we have not developed an algorithm to translate UnQL queries into CTL formulae, we can conclude that our approach allows to correctly deal with join-free queries of UnQL.

GraphLog is a query language based on a graph representation of both data and queries [12]. Queries represent graph patterns corresponding to paths in databases. For example, the graph I in Fig. 6 is a representation of a flights schedule database. Each flight number is related to the cities it connects (by the predicates *from* and *to*, respectively) and to the departure and arrival time (by the predicates *departure* and *arrival*).

GraphLog represents databases by means of *directed labeled multigraphs* which precisely correspond to $\mathbb{W}$-instances; these databases are not required to be rooted and therefore the corresponding $\mathbb{W}$-graphs are not necessarily rooted. Queries ask for patterns that must be present or absent in the database graph. A graphical query is a set of query graphs, each of them defining (constructive part) a set of new edges (relations) that are added to the graph whenever the path (non-constructive part) specified in the query itself is found (or not found for negative requests). More in detail, a query graph is a directed labeled multigraph with a distinguished edge (it defines a new relation that will hold, after the query application, between two objects in a chosen database whenever they fulfill the requirements of the query graph) and without isolated nodes. Moreover, each edge is labeled by a literal (i.e. positive or negative occurrence of a predicate applied to a sequence of variables and constants) or by a *closure literal*, which is simply a literal followed by the positive closure operator; each distinguished edge can only be labeled by a positive non-closure literal.

For example, graph (G) in Fig. 6 requires to find in a database two flights $F1$ and $F2$ departing from and arriving to the same city C, respectively. The edge 'result' will connect $F1$ and $F2$. The CTL formula associated with the non-costructive part of this query is:

$$\Psi(G) = \mathsf{Flight} \wedge \mathsf{EX}_{\mathsf{to}}(\mathsf{City} \wedge \mathsf{EX}_{\mathsf{from}^{-1}}\mathsf{Flight})$$

(which describes the node $F1$). $\Psi(G)$ is satisfied by the state labeled 109.

The possibility to express closure literals causes some difficulties in the translation of query graphs into CTL formulae. Consider for example, the two query graphs in Fig. 7 representing relationships between people. Their natural translation into CTL should be:

$\Psi(Q_1) = \mathsf{Person} \wedge (\mathsf{EX}_{\mathsf{parent}}\mathsf{Person}) \rightarrow (\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{parent}}(\mathsf{Person} \wedge$
$\mathsf{EX}_{\mathsf{relative}^{-1}}\mathsf{Person}))$

$\Psi(Q_2) = \mathsf{Person} \wedge (\mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}\mathsf{Person})) \rightarrow$
$(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}^{-1}}\mathsf{Person})))$

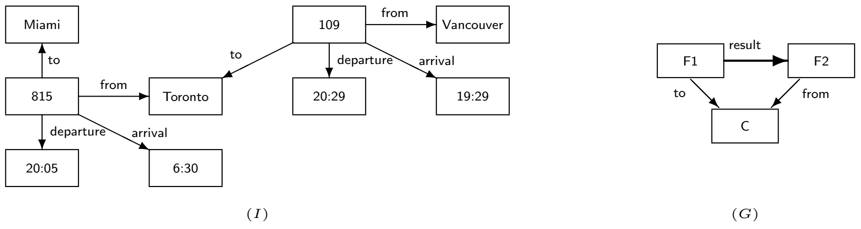


Fig. 6. GraphLog representation of a database and a query

It is easy to see that the node n of the instance graph $\mathcal{I}$ in Fig. 7 satisfies the formula $\Psi(Q_1)$ but does not fulfill the natural interpretation of relationship between people. The problem is that in CTL it is not possible to constrain that the last unary predicate Person in $\Psi(Q_1)$ has to be satisfied by the same state of the first unary predicate Person.

In order to overcome this limitation one should firstly compute the closure of labeled graphs representing databases. Intuitively, computing graph-theoretical closure entails inserting a new edge labeled a between two nodes, if they are connected via a path of any length composed of edges that are all labeled a in the original graph (see [14] for an application of graph closure to XML documents). Computing the closure is well-known to be polynomial time w.r.t. the size of nodes of the graph.

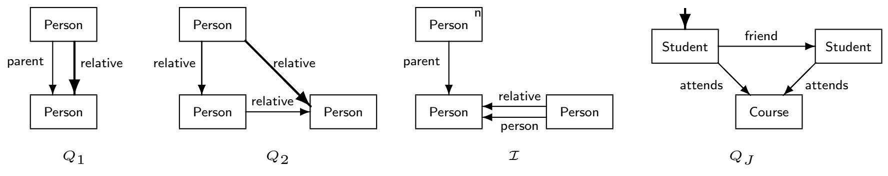


Fig. 7. Two query graphs, an instance, an a 'join' query

We can conclude that our approach based on model-checking techniques allows to correctly deal with join-free queries of GraphLog without closure operators.

G-Log [27], as GraphLog, is a query language in which data, queries, and generic rules are expressed by graphs. As far as instances are concerned, syntax of G-Log instances is the same as that of $\mathbb{W}$. G-Log queries allow more flexibility then $\mathbb{W}$-ones. Nevertheless, often G-Log queries has only one "green" node with an edge pointing to a "red" node ν. These queries are the same as $\mathbb{W}$-queries, in which ν is the node pointed by the query. In G-Log two kinds of nodes are allowed, complex and atomic nodes, but this feauture can be simulated by modifying the co-domain of the *label* function. Instead, solid edges are forbidden to enter into dashed nodes in G-Log. This feauture of $\mathbb{W}$ is one of the main points for programming nesting quantification and for expressing in a compact way the universal quantification. Only existential requirements are instead allowed in G-Log queries. If a $\mathbb{W}$ query fulfills this further requirement, is a G-Log query.

Semantics of query application is given in [27] via the notion of *graph embedding* and in [13] (cf. [17]) using the notion of *subgraph bisimulation* (both NP complete). We have formally proved that for acyclic queries that are allowed both in $\mathbb{W}$ and in G-log (with the semantics of [13]) the results of query application is the same in the two languages. We have already discussed (Sect. 6) that for complexity limits we cannot extend the method to all cyclic queries.

The problem of expressing *join* conditions by means of *CTL* formulae remains open, because in the propositional modal logics framework there is no way to distinguish different states having the same set of local properties. A possibility to overcome this problem is to add to the set of local properties of each state an identificator of the state (of course, we assume that the set of all states is finite). For example, the $\mathbb{W}$-query Q_J in Fig. 7 requires to find all the Students that attend a Course together with (at least) one friend. If we know that the identificators of Course nodes are c_1 and c_2, we could translate the query with the *CTL* formula

$$(\mathsf{Student} \wedge \mathsf{EX}_{\mathsf{attends}}(\mathsf{Course} \wedge c_1) \wedge \mathsf{EX}_{\mathsf{friend}}(\mathsf{Student} \wedge \mathsf{EX}_{\mathsf{attends}}(\mathsf{Course} \wedge c_1))) \vee$$
$$(\mathsf{Student} \wedge \mathsf{EX}_{\mathsf{attends}}(\mathsf{Course} \wedge c_2) \wedge \mathsf{EX}_{\mathsf{friend}}(\mathsf{Student} \wedge \mathsf{EX}_{\mathsf{attends}}(\mathsf{Course} \wedge c_2)))$$

This translation causes an explosion of the formula size and it requires also to know in advance the set of identificators of a certain state (in the example the Course state).

8 Conclusions and Future Work

In this work we have shown how it is possible to effectively solve the data retrieval problem for semistructured data using techniques and algorithms coming from the model-checking community. Note that this approach could also be applied to XML-based languages such as Quilt [8] and XPath [29]. The input and output of these languages are documents usually modeled as trees. Once again, in this setting model-checking algorithms can correctly deal with properties expressing conditions to be satisfied on paths but without joins, grouping, or aggregation functions of SQL.

Some other works on this direction are [9, 23, 24]. In [9] the author proposes an approach for inferring properties of models by using a model checker. He extends the syntax of *CTL* by allowing the special symbol '?', and characterizes a subset of this new language that allows linear-time decidability. In [23] the authors suggest to query data using 'query automata', that are particular cases of two-ways deterministic tree automata. They show the equivalence with formula expressible in second-order monadic logic. In [24] they identifies subclasses of formulae/automata that allow efficient algorithms. A future work is to determine the exact relationships with our approach.

The *graph-formula* translation could be extended to a wider family of graphs including join conditions, and it might be interesting to find the exact subset of *CTL** (whose model-checking problem is PSPACE complete) needed to translate queries of interest. Moreover, similar techniques are shown to be possible for extracting formulae from SQL-like queries. The modal logic semantics forces a bisimulation equivalence between subgraphs fulfilling the same formula. This allows us to say that two nodes with the same properties are equivalent but not to require that they are the same node. This prevents us to model correctly the *join* operation. We have implemented the method on the model-checker NuSMV. However, for an effective implementation on Web-databases some form of heuristic (e.g. check part of the domain name) must be applied in order to cut the search space for accessible data (that can be all the web). This issue is studied in [15] in a slight different context. Another future direction is to mix this technique with constraint-based data retrieval (where temporal formulae can be used directly).

Acknowledgements

Nico Lavarini has contributed with his Master's Thesis to the earlier stages of the research presented in this work. We thank also A. Cimatti, R. Giacobazzi, C. Piazza, M. Roveri, and L. Tanca for useful discussions.

References

1. S. Abiteboul. Querying semi-structured data. In *Proc. of ICDT*. Vol. 1186 of *LNCS*, pp. 1–18, 1997.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int'l J. on Digital Libraries*, 1(1):68–88, 1997.
3. P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proc. of the 1996 ACM SIGMOD*, pp. 505–516, 1996.
4. P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. Adding structure to unstructured data. In Proc. of *Database Theory; 6th Int'l Conf.*, pp. 336–350, 1997.
5. P. J. Cameron. First-Order Logic. In L. W. Beineke and R. J. Wilson (Eds.): *Graph Connections. Relationships Between Graph Theory and other Areas of Mathematics.* Clarendon Press, 1997.
6. L. Cardelli and G. Ghelli. A query language for semistructured data based on the Ambient logic. In Proc. of *ESOP 2001*, Vol. 2028 of LNCS, pp. 1–22, 2001.
7. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a graphical language for querying and restructuring XML documents. *Proc. of WWW8*, Canada, 1999.
8. D. Chamberlin, J. Rubie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *Proc. of the World Wide Web and Databases, Third International Workshop WebDB 2000*, Vol. 1997 of LNCS, pp. 1–25, 2001.

9. W. Chan. Temporal-logic Queries. In *Proc. of 12th CAV*. Vol. 1855 of LNCS, pp. 450–463. Chicago, USA, 2000.
10. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. *Proc. of 11th CAV*. Vol. 1633 of LNCS, pp. 495–499, 1999.
11. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specification. *ACM TOPLAS*, 8(2):244–263, 1986.
12. M. P. Consens and A. O. Mendelzon. GraphLog: a Visual Formalism for Real Life Recursion. In Proc. of the 9th *ACM PODS'90*, pp. 404–416, 1990.
13. A. Cortesi, A. Dovier, E. Quintarelli, and L. Tanca. Operational and Abstract Semantics of a Query Language for Semi-Structured Information. In *Proc. of DDLP'98*, pp. 127–139. GMD Report 22, 1998. Extended Version to appear in *Theoretical Computer Science*.
14. E. Damiani and L. Tanca. Blind Queries to XML Data. In *Proc. of 11th International Conference, DEXA 2000*, Vol. 1873 of LNCS, pp. 345–356, 2000.
15. L. de Alfaro. Model Checking the World Wide Web. In *Proc. of 13th Conference on Computer Aided Verification*, Vol. 2102 of LNCS, pp. 337–349, 2001.
16. A. Dovier and C. Piazza. The Subgraph Bisimulation Problem and its Complexity. Univ. di Udine, Dip. di Matematica e Informatica, RR 27/00, Nov. 2000.
17. A. Dovier and E. Quintarelli. Model-Checking Based Data Retrieval. Technical Report, Politecnico di Milano, May 2000 (www.elet.polimi.it/∼quintare).
18. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B: Formal Models and Semantics. Elsevier, and MIT Press, 1990.
19. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.
20. P. C. Kannellakis and S. A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86(1):43–68, 1990.
21. R. Milner. A Calculus of Communicating Systems. Vol. 92 of LNCS, 1980.
22. M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking. A tutorial introduction. In *Proc. of SAS'99*. Vol. 1694 of LNCS, pp. 330–354, 1999.
23. F. Neven and T. Schwentick. Query Automata. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. of DB Systems*, ACM Press, pp. 205–214, 1999.
24. F. Neven and T. Schwentick. Expressive and Efficient Pattern Languages for Tree-Structured Data. In *Proc. of the 19th ACM SIGACT-SIGMOD-SIGART Symp. on Princ. of DB Systems*, ACM Press, pp. 145–156, 2000.
25. R. Paige and R. E. Tarjan. Three Partition refinements algorithms. *SIAM J. on Computing*, 16(6):973–989, 1987.
26. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of the 11th ICDE*, pp. 251–260, 1995.
27. J. Paredaens, P. Peelman, and L. Tanca. G–Log: A Declarative Graphical Query Language. *IEEE TKDE*, 7(3):436–453, 1995.
28. D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying Semistructured Heterogeneus Information. In *Proc. of DOOD'95*, pp. 319–344, 1995.
29. World Wide Web Consortium. *XML Path Language (XPath) version 1.0.* www.w3.org/TR/xpath.html, W3C Reccomendation, November 1999.