Applying Model-Checking to solve Queries on Semistructured Data

A. Dovier* E. Quintarelli†

April 28, 2006

Abstract

The large volume and nature of data available to the casual users and programs motivate the increasing interest of the database community in studying flexible and efficient techniques for extracting and querying semistructured data. On the other hand, efficient methods have been discovered for solving the so-called model-checking problem for some modal logics. The aim of this paper is to show how some of these methods can be used for querying semistructured data. For doing that we show that semistructured data can be naturally seen as Kripke Transition Systems. To keep the presentation independent of a specific language, we introduce a graphical query language that includes some of the features of the query languages based on graphs and patterns. We show how to associate *CTL* formulas to queries of this language. This allows us to see the problems of solving a query as an instance of the model-checking problem for *CTL* that can be solved in polynomial time. We have tested the method by using a model-checker, and have studied the applicability of the method to some existing languages for semistructured databases.

Keywords. Semistructured Databases, *CTL*, Model-Checking.

1 Introduction

In recent years the database research community has concentrated on the introduction of methods for representing and querying *semistructured data*, since most of the information accessible through Internet is typically semistructured. This kind of data has no absolute schema fixed in advance, and its structure may be irregular or incomplete [1].

It is a common approach to represent semistructured data by using data models based on directed labeled graphs [2, 9, 11, 38], thus, a query is expected to extract information stored in labeled graphs; the data retrieval activity can be reduced to the problem of finding subgraphs of the database instance graph that satisfy the requirements of the query. In some of these languages, queries are graphs themselves [15, 23, 36], therefore the above activity amounts to find subgraphs of the instance graph that *match* the graph representing the query. In other cases, queries are written in extended SQL languages [3, 8, 10]; however, when the semantics of a query is understood, it may be rewritten as a graph and data retrieval can be performed as above.

*Dip. di Matematica e Informatica, Univ. di Udine. Via delle Scienze 206, 33100 Udine (Italy). email: dovier@dimi.uniud.it

†Dip. di Elettronica e Informazione, Politecnico di Milano. Piazza Leonardo da Vinci 32, 20133 Milano (Italy). email: quintare@elet.polimi.it

Let us focus on the problem of defining graph *matching* or *equivalence*. Two main graph-equivalence relations are graph *isomorphism* and graph *bisimulation*. Isomorphism is a syntactic notion and there are neither polynomial time algorithms for the Subgraph Isomorphism problem (NP-complete) nor for the Graph Isomorphism problem (which is in NP but currently it is not known whether it is in P or not). The semantic notion of bisimulation [32, 37] seems to better fit the data retrieval activity [8, 16], and, moreover, the bisimulation test between two graphs can be done in polynomial time [19, 29, 35]. Thus, it is reasonable to choose this relation as the main notion for verifying graph equivalence. For example, in [9] the authors use bisimulation for defining equivalence between graph-based instances of semistructured documents. In [16] the notion of bisimulation is used to formalize the concept of similarity between graphs representing queries and instances; in particular, the application of a query to an instance requires to solve the task of finding subgraphs of the instance that are bisimilar to the query. Since the task of finding subgraphs bisimilar to a given one is NP-complete [20], the full solution to this problem is not feasible.

The main aim of this work is to show the connection between query languages applied to semistructured data and modal logics, and to characterize families of queries that admit efficient solutions to the data retrieval problem. The novel ideas are to associate a *modal* logic formula Ψ to a graphical query (or constraint), 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 traverse in order to reach the required information. Let us observe however that we are not dealing with temporal databases, since time does not appear in the semantics of our formulae.

This way, finding subgraphs of the database instance graph that match the query is reduced to the problem of finding nodes of the KTS derived from the database instance graph that satisfy Ψ. This is an instance of the *model-checking* problem, and it is well-known that if the formula Ψ belongs to the class of *CTL* formulae, then the problem is decidable in linear time in the product of the sizes of the KTS and the formula [14] (this is the so-called *combined complexity* [39]).

We identify some families of graph-based queries that correspond to *CTL* formulae. For doing that, we abstract from each specific existing language. We use a "toy" query language called $\mathbb{W}$ that includes some of the features of several languages in which queries are graphical or based on patterns, such as Lorel [3], G-Log [36], Graphlog [15], UnQL [23], and XPath [41]. As an immediate consequence, an effective procedure for efficiently querying semistructured databases can be directly implemented on a model-checker. Then we relate the results in $\mathbb{W}$ to these languages and show that the translation allows a polynomial time implementation of parts of these languages. We define $\mathbb{W}$not to introduce a new query language with a full expressive power, but to determine the common fragment, which is part of all existing query languages, that can be efficiently managed by using model-checking algorithms. In our opinion, this fragment includes queries that are frequent in practice and can be expressed by the languages XPath, UnQL, Graphlog. This opinion has been recently enforced by encodings of XPath queries into model checking formulas [26, 30, 5].

While we have effectively tested the approach by using the model-checker NuSMV [13], any other tool for *CTL* model-checking could be used, as well. A preliminary version of this paper was presented in [21].

The structure of this paper is as follows. Sections 2 and 3 introduce basic definitions of the model-checking problem and of the language $\mathbb{W}$. In Section 4 we explain the graph-

based semantics of the $\mathbb{W}$-language, in Section 5 we show how to assign a KTS to a database instance, while in Section 6 it is shown how to extract CTL formulae from query graphs. In Section 7 we report some basic results on the complexity of the data retrieval problem by using our approach. A complete example with experimental results is given in Section 8, while Section 9 is devoted to applying the method to other languages. Related work is discussed in Section 12. Some conclusions are sketched in Section 13.

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 [22].

Definition 2.1 *A* Kripke Transition System *(KTS) over a set* Π *of* atomic propositions *is a tuple* $\mathcal{K} = \langle \Sigma, \mathcal{A}ct, \mathcal{R}, I \rangle$*, where* Σ *is a set of states,* $\mathcal{A}ct$ *is a set of actions (*$\Pi \cap \mathcal{A}ct = \emptyset$*),* $\mathcal{R} \subseteq \Sigma \times \mathcal{A}ct \times \Sigma$ *is a total transition relation, and* $I : \Sigma \to \wp(\Pi)$ *is an interpretation.*

Definition 2.2 *Given* Π *and* $\mathcal{A}ct$*,* 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, and* $a \subseteq \mathcal{A}ct$*, 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.*

A and E are the universal and existential path quantifiers, while X (neXt) and U (Until) are the time modalities. Composition of formulae of the form $\varphi_1 \vee \varphi_2$, $\varphi_1 \to \varphi_2$, 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. [22]). When a is of the form $\{m\}$ for a single action m, we simplify the notation writing $\mathsf{AX}_m, \mathsf{EX}_m, \mathsf{AX}_m, \mathsf{EX}_m$.

A *path* (called fullpath in [22]) 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 $j > i$ it holds that a_{j-1} is the special action $\circlearrowleft$ which is not in $\mathcal{A}ct$ and $\pi_j = \pi_i$.

Definition 2.3 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* $s \not\models \phi$*;*
- $s \models \phi_1 \wedge \phi_2$ *iff* $s \models \phi_1$ *and* $s \models \phi_2$*;*
- $s \models \mathsf{EX}_a(\phi)$ $(a \subseteq \mathcal{A}ct)$ *iff there is a path* $\pi = \langle s, x, \pi_1, \ldots \rangle$ *s.t.* $x \in a$ *and* $\pi_1 \models \phi$*;*
- $s \models \mathsf{AX}_a(\phi)$ $(a \subseteq \mathcal{A}ct)$ *iff for all paths* $\pi = \langle s, x, \pi_1, \ldots \rangle$ *s.t.* $x \in a$*, it holds that* $\pi_1 \models \phi$*;*

- $s \models \mathsf{EU}_a(\phi_1, \phi_2)$ $(a \subseteq \mathcal{A}ct)$ *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* $\ell_i \in a)$;
- $s \models \mathsf{AU}_a(\phi_1, \phi_2)$ $(a \subseteq \mathcal{A}ct)$ *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 2.4 *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.* $s \models \varphi$.

The above problems are usually formulated for Σ finite. In this case, the global model-checking problem for a *CTL* formula φ can be solved in linear running time on $|\varphi| \cdot (|\Sigma| + |\mathcal{R}|)$ [14].

Remark 2.5 *KTS are very expressive structures modeling modal logics. Kripke Structures (KS) are KTS in which transitions are not driven by actions. Some model-checkers (e.g., NuSMV [13]) are based on KS. Standard techniques can be used to transform KTS in KS [33]. The idea is to store the action of a transition in the reached state. Formally, given a KTS* $\mathcal{K} = \langle \Sigma, \mathcal{A}ct, \mathcal{R}, I \rangle$ *over the set* Π *of atomic propositions,* $\mathcal{K}_T$ *is the Kripke Structure* $\langle \Sigma \times \mathcal{A}ct, \mathcal{R}', I' \rangle$ *over* $\Pi \cup \mathcal{A}ct$, *with* $\mathcal{R}' = \{(\langle s, a \rangle, \langle s', a' \rangle) \mid \exists a' \exists s' (s \xrightarrow{a'} s') \in \mathcal{R}\}$ *and* $I'(\langle s, a \rangle) = I(s) \cup \{a\}$.

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

In this section we describe the syntax of the language $\mathbb{W}$, a simple graph-based language that we use to express instances and queries.

Definition 3.1 *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* $\langle m, attribute, n \rangle$, ℓ *is a function* $\ell : N \longrightarrow \mathcal{C} \times (\mathcal{L} \cup \{\bot\})$, $\mathcal{C} = \{$ *solid, dashed* $\}$, *and* $\mathcal{L}$ *is a set of labels.*

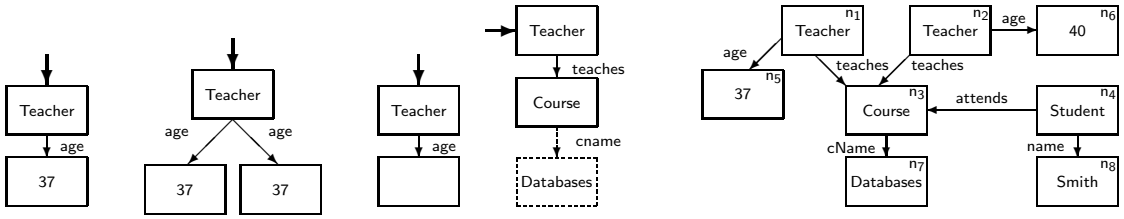


Figure 1: Five $\mathbb{W}$-graphs: four $\mathbb{W}$-queries and a $\mathbb{W}$-instance I

In Fig. 1 five $\mathbb{W}$-graphs are depicted (please, ignore thick arrows, whose meaning will be given in Def. 3.2). Edge attributes are made by two components, the *color* and the *label*. The label is explicitly written close to the corresponding edge. The function ℓ returns a *color* and a (possibly empty) *label* for each node. Node labels are written inside the rectangles representing the nodes. The empty label $\bot$ means 'undefined' (or *dummy*). Graphically, it is represented by omitting the label (see, e.g., the third graph from the left). The set of colors $\mathcal{C}$ denotes how the lines of nodes and edges are drawn. We also call this

information the *color* of a node/edge. ℓ can be seen as the composition of the two single-valued functions $\ell_{\mathcal{C}}$ and $\ell_{\mathcal{L}}$. ℓ is implicitly defined also on 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 attributes are different. Given two sets $S, T \subseteq N$, T is *accessible* from S if for each node $n \in T$ there is a node $m \in S$ such that there is a directed path in G from m to n. For instance, in the rightmost graph of Fig. 1, the set $\{n_3, n_5, n_6, n_7, n_8\}$ is accessible from the set $\{n_1, n_2, n_4\}$. Let us observe that node names $n_1, \ldots, n_8$ are identifiers used to refer nodes in the example; they are not part of the $\mathbb{W}$-graph (however, the generality of $\mathcal{L}$ would allow labels made of two parts, one of which is the node number).

$\mathbb{W}$-graphs can be of two types: instances and queries.

Definition 3.2 *A $\mathbb{W}$-instance I is a $\mathbb{W}$-graph such that $\ell_{\mathcal{C}}(e) =$ solid for each edge e of I and $\ell_{\mathcal{C}}(n) =$ solid and $\ell_{\mathcal{L}}(n) \neq \perp$ for each node n of I.*

A $\mathbb{W}$-query *is a pointed $\mathbb{W}$-graph $\langle G, \nu\rangle$ with $\ell_{\mathcal{C}}(\nu) =$ solid, i.e. ν is a solid 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 $\{\nu\}$. If for all nodes n and edges e of G it holds that $\ell_{\mathcal{C}}(n) = \ell_{\mathcal{C}}(e) =$ solid then $\langle G, \nu\rangle$ is said a* solid $\mathbb{W}$-query*.*

The thick arrows of the four leftmost graph in Fig. 1 are their points. Thus, they are $\mathbb{W}$-queries. Intuitively, their semantics (from the leftmost to the rightmost) is the following: collect all the Teachers whose age is 37 (for the first two queries, the second is, in some sense, redundant), collect all the Teachers with *some* age, and collect all the Teachers that teaches some Course, save those that teach Databases. The rightmost $\mathbb{W}$-graph is an instance and describes semi-structured information (two teachers, one 37 years old, one 40 years old, both of them teach Databases, and the student Smith attend the same course).

We assume the standard notions of (directed) path and of subgraph. We denote as $G' \sqsubseteq G$ the fact that G' is a subgraph of G. $G' \sqsubset G$ states that $G' \sqsubseteq G$ and G and G' are different. The *size* of of a graph $G = \langle N, E, \ell\rangle$ is $|G| = |N| + |E|$. Moreover, if $G = \langle N, E, \ell\rangle$ is a $\mathbb{W}$-graph, then $G_s = \langle N_s, E_s, \ell|_{N_s}\rangle$ is the solid subgraph of G, i.e.: $N_s = \{n \in N : \ell_{\mathcal{C}}(n) = \text{solid}\}$ and $E_s = \{\langle m, \langle \text{solid}, \ell\rangle, n\rangle \in E : m, n \in N_s\}$.

4 Bisimulation-Based Semantics of a subset of $\mathbb{W}$

We provide a bisimulation-based semantics for simple $\mathbb{W}$-queries. For these queries the language $\mathbb{W}$ is the same as the graphical query language G-log [36]; a bisimulation-based semantics for G-log is described in [16].

We start by recalling the notion of *bisimulation* [37, 32].

Definition 4.1 *Given two $\mathbb{W}$-graphs $G_0 = \langle N_0, E_0, \ell^0\rangle$ and $G_1 = \langle N_1, E_1, \ell^1\rangle$, a relation $b \subseteq N_0 \times N_1$ is said to be a* bisimulation *between G_0 and G_1 if and only if:*

1. *for $i = 0, 1$, $(\forall n_i \in N_i)(\exists n_{1-i} \in N_{1-i}) n_0\ b\ n_1$,*
2. *$(\forall n_0 \in N_0)(\forall n_1 \in N_1)(n_0\ b\ n_1 \rightarrow (\ell^0_{\mathcal{L}}(n_0) = \ell^1_{\mathcal{L}}(n_1) \vee \ell^0_{\mathcal{L}}(n_0) = \perp \vee \ell^1_{\mathcal{L}}(n_1) = \perp))$, and*
3. *for $i = 0, 1$, $(\forall n \in N_i)$, let $M_i(n) =_{def} \{\langle m, label\rangle : \langle n, \langle color, label\rangle, m\rangle \in E_i\}$.*
   Then, $(\forall n_0 \in N_0)(\forall n_1 \in N_1)$ such that $n_0\ b\ n_1$, for $i = 0, 1$ it holds that

$$(\forall \langle m_i, \ell_i\rangle \in M_i(n_i))(\exists \langle m_{1-i}, \ell_{1-i}\rangle \in M_{1-i}(n_{1-i}))(m_0\ b\ m_1 \wedge \ell_i = \ell_{1-i})\,.$$

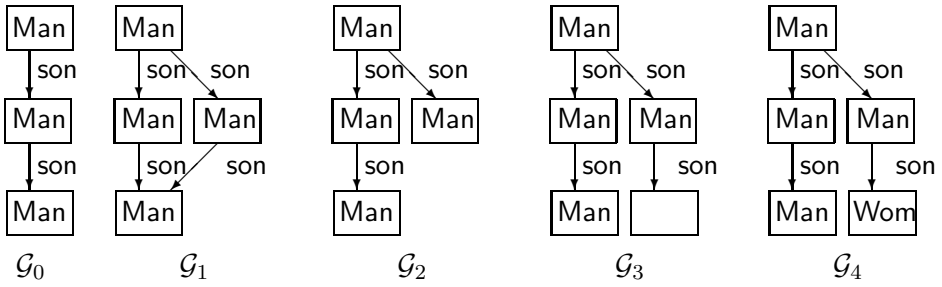


Figure 2: Bisimilar and not bisimilar acyclic $\mathbb{W}$-graphs

We write $G_0 \overset{b}{\sim} G_1$ ($G_0 \overset{b}{\not\sim} G_1$) if b is (not) a bisimulation between G_0 and G_1. We write $G_0 \sim G_1$ ($G_0 \not\sim G_1$) if there is (not) a bisimulation between G_0 and G_1: in this case we also say that G_0 is bisimilar to G_1.

Condition (1) states that no node in the two graphs can be left out of the relation b. Condition (2) states that two nodes belonging to relation b have exactly the same label, save for the case of dummy nodes, labeled by $\perp$ (this is not considered in the standard definition). Finally, condition (3) deals with edge correspondences. If two nodes n_0, n_1 are in relation b, then every edge having n_0 as endpoint should find as a counterpart a corresponding edge with n_1 as endpoint. Notice that output values of the $\ell_{\mathcal{C}}$ function (solid/dashed) are not taken into account in the bisimulation definition.

Consider, as an example, the graphs in Figure 2. Graphs $\mathcal{G}_0$ and $\mathcal{G}_1$ are bisimilar. None of them is bisimilar to $\mathcal{G}_2$. $\mathcal{G}_2$ is not bisimilar to any other graph. The "dummy" node of graph $\mathcal{G}_3$ allows it to be bisimilar to $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_4$. Actually, allowing dummy nodes to be in relation with labeled ones destroys the transitivity property of classical bisimulation relation. As a matter of fact, $\mathcal{G}_1 \sim \mathcal{G}_3$, $\mathcal{G}_3 \sim \mathcal{G}_4$, but $\mathcal{G}_1 \not\sim \mathcal{G}_4$. The three leftmost graphs of Figure 1 are bisimilar. However, we will look for bisimilarity of queries (possibly with dummy nodes) with parts of instance graphs (where dummy nodes are not allowed). We do not need of reasoning about the bisimilarity of queries.

Definition 4.2 *Let I be a $\mathbb{W}$-instance and C a $\mathbb{W}$-graph. Let $I' \sqsubseteq I$ such that $I' \overset{b}{\sim} C$. I' is* maximal *if there is no I'' s.t. $I' \sqsubset I'' \sqsubseteq I$ s.t. $I'' \overset{b'}{\sim} C$ and $b \subseteq b'$.*

We give the bisimulation-based semantics of solid $\mathbb{W}$-queries (Def. 3.2).

Definition 4.3 *Let I be a $\mathbb{W}$-instance and Q a solid $\mathbb{W}$-query. The semantics of Q w.r.t. I is the graph $C \sqsubseteq I$ such that $C \overset{b}{\sim} Q$ and C is maximal.*

In practice, we are interested in the set of nodes of I that are in relation b with the point ν of the query $Q = \langle G, \nu \rangle$. A graph of the kind above may not exists. In this case, the output is the empty graph.

As an example, let us consider the three leftmost queries and the instance I of Figure 1. The intuitive meaning of the first one is: collect all the teachers aged 37. Its semantics on the instance I is to return the subgraph made of nodes n_1 and n_5, with n_1 bisimilar to the point of the query. The second query is equivalent to the first one, but it is redundant. Its bisimulation-based semantics is the same. The intuitive semantics of the third query is that of asking for all the teachers that have declared some age (see also Fig. 3). The graph

made by nodes n_1, n_5, n_2, n_6 is returned, with n_1 and n_2 in relation with the point. The subgraph made by nodes n_1 and n_2 is also bisimilar but it is not maximal. Observe the use of the dummy node labeled $\perp$ allows us to put both n_2 and n_6 in relation with the lowest node of the query.

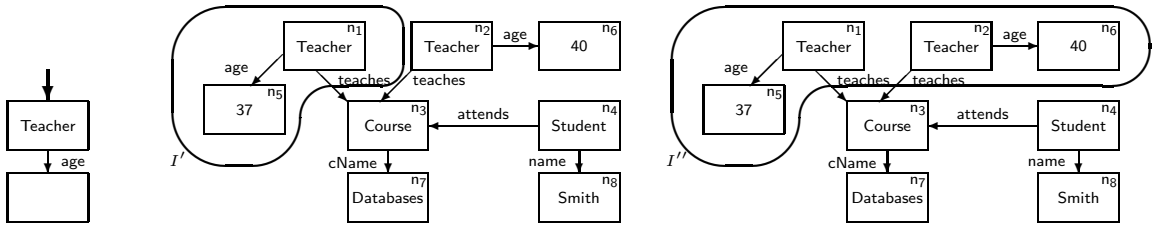


Figure 3: A query Q an instance, and two subgraphs bisimilar to Q. I'' is maximal.

Remark 4.4 *Let us conclude the section by pointing out a computational limit of any language whose semantics is based on Def.* 4.3. *We know from [20] that establishing whether there is a subgraph C of a graph I that is bisimilar to a given graph Q (Subgraph Bisimulation Problem) is NP-complete. This means that there are families of solid $\mathbb{W}$-queries for which there is no polynomial time algorithm for retrieving the data (unless $P = NP$). The rest of the paper is devoted to define, instead, families of queries for which polynomial time techniques exist.*

5 $\mathbb{W}$-Instances as KTS

In this section we show how to associate a *KTS* to a $\mathbb{W}$-instance.

Definition 5.1 *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 $\Pi_G = \{p : (\exists n \in N)(p = \ell_{\mathcal{L}}(n))\}$ (the set of all node labels of G) as follows:*

- *The set of states is $\Sigma_G = N$.*
- *The set of actions $\mathcal{A}ct_G$ includes all the edge labels. In order to allow to traverse each edge in the two directions, we also add in $\mathcal{A}ct_G$ actions for the inverse relations. To model negation, actions for the negation of all the relations are introduced. Thus, if $m \xrightarrow{p} n$ belongs to E we add the actions $p, p^{-1}, \bar{p}, \bar{p}^{-1}$. Let $\mathcal{A}ct_G^+ = \{p, p^{-1} : (\exists m \in N)(\exists n \in N)(\langle m, p, n \rangle \in E)\}$. We define $\mathcal{A}ct_G = \{q, \bar{q} : q \in \mathcal{A}ct_G^+\}$.*
- *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 $\mathcal{R}_G = \tilde{E}_G \cup \{\langle m, \bar{p}, n \rangle : m, n \in N, p \in \mathcal{A}ct_G^+, \langle m, p, n \rangle \notin \tilde{E}_G\}$.*
- *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.[1] *$\mathcal{I}_G(n) = \{\mathsf{true}, \ell_{\mathcal{L}}(n)\}$.*

[1]The two unary atoms represent the basic local properties of a system state. Other properties can be added, if needed.

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 $q = p^{-1}$, $q \in \Pi_G$.

Model-checking procedures also require a set B_G of initial states for $\mathcal{K}_G$ such that the set Σ_G of all the states of $\mathcal{K}_G$ is accessible from B_G. Semistructured data typically have a set of entry points (e.g., the unique root).

Example 5.2 *Consider the graph $G = \langle \{n_1, \ldots, n_8\}, E\rangle$, of Fig. 1. Then:*

- $\Pi_G = \{$ Teacher, Course, Student, 37, 40, Databases, Smith$\}$
- $\Sigma_G = \{n_1, \ldots, n_8\}$; $B_G = \{n_1, n_2, n_4\}$.
- $\mathcal{A}ct_G = \{$ teaches, teaches^{-1}, $\overline{\text{teaches}}$, $\overline{\text{teaches}^{-1}}$, attends, $\ldots$ age, $\ldots$, cName, $\ldots$, name, $\ldots\}$
  $\mathcal{A}ct_G^+ = \{\text{teaches}, \ldots, \text{name}, \text{teaches}^{-1}, \ldots, \text{name}^{-1}\}$.
- $\tilde{E}_G$ *is the union of E with the set*

$$\begin{array}{ll}\{ & \langle n_3, \text{ teaches}^{-1}, n_1\rangle, \langle n_3, \text{teaches}^{-1}, n_2\rangle, \langle n_3, \text{attends}^{-1}, n_4\rangle, \\ & \langle n_5, \text{age}^{-1}, n_1\rangle, \langle n_6, \text{age}^{-1}, n_2\rangle, \langle n_7, \text{cName}^{-1}, n_3\rangle, \langle n_8, \text{name}^{-1}, n_4\rangle\}\end{array}$$

- *Then $\mathcal{R}_G$ is the union of $\tilde{E}_G$ with the "complement" set:*

$$\{\langle m, \bar{p}, n\rangle \,:\, m, n \in \Sigma_G, p \in \mathcal{A}ct_G^+ \wedge \langle m, p, n\rangle \notin \tilde{E}_G\}$$

- *The interpretation $\mathcal{I}$ is:*

$$\mathcal{I}(n_1) = \{\mathsf{true}, \text{Teacher}\}, \mathcal{I}(n_2) = \{\mathsf{true}, \text{Teacher}\}, \ldots, \mathcal{I}(n_8) = \{\mathsf{true}, \text{Smith}\}$$

Remark 5.3 *The definition of path in Section 2 uses the notion of self loop $\circlearrowleft$ for leaf nodes of a KTS. We do not need to deal with self loops, since the presence of inverse and complemented edges ensure that there are no leaves in $\mathcal{K}_G$.*

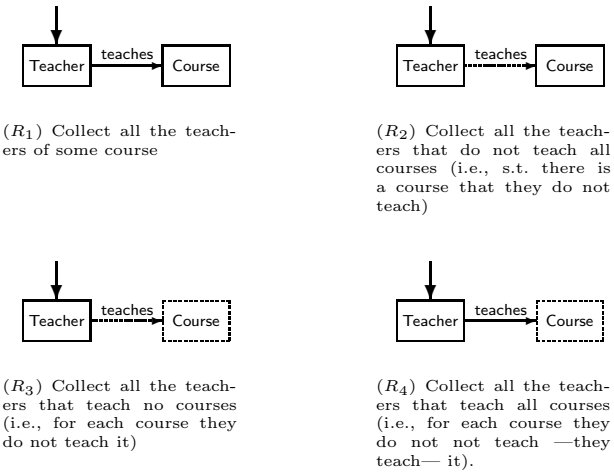
6 CTL-based semantics of $\mathbb{W}$

In this section we define a temporal logic semantics to $\mathbb{W}$-queries, by associating a formula $\Psi_\nu(G)$ to a query $\langle G, \nu\rangle$. Such a formula will give us the possibility to define a model-checking based methodology for querying a $\mathbb{W}$-instance. We remark that in our interpretation two nodes connected by an arrow are not two states ordered with respect to time, but two nodes linked by any kind of relationship.

We anticipate this definition in order to clarify the meaning of formula encoding.

Definition 6.1 (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 $s \models \Psi_\nu(G)$.*

In Section 4 we have not discussed the meaning of dashed nodes and edges in queries. Basically, they allow us to express negative requirements. We deal with them in this section.



(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).

Figure 4: Simple queries

6.1 Technique Overview

We start our analysis by considering simple queries whose graphs consist of two nodes (Fig. 4). Query R_1 has no dashed part. 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 directed path on the graph, nesting formulae that must be satisfied by one (EX) or every (AX) next state.

Query R_2 contains a dashed edge teaches that expresses a negative requirement. 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 teaches, allows us to express the same concept by requiring that, between the two 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 Teacher s.t., for all the nodes labeled Course, the relation teaches is not satisfied". 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 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 intuitive meaning is: "it is true if Teacher is linked by edges labeled teaches to all the Course nodes of the graph". Note the extremely compact way for graphically expressing universal quantification.

Remark 6.2 *From rules* $(R_1), (R_2), (R_3)$ *one can obtain corresponding G-log rules by adding* green *nodes as origins of the edges (thick arrows) which are the points of the queries. Their G-log non-constructive semantics is consistent with the CTL-based semantics proposed here (cf. also Section* 9.4*). Rule* (R_4)*, instead, does not have a counter-part in the language G-log.*

6.2 Query Translation

We will show how to encode $\mathbb{W}$-queries in *CTL* formulae. As explained in Remark 4.4, we know that we cannot encode all possible queries in a framework that can be solved in polynomial time. We will focus on four families of queries, three of them acyclic.

As initial step, we associate a formula φ to each node and edge of a graph G.

Definition 6.3 *Let* $G = \langle N, E, \ell \rangle$ *be a* $\mathbb{W}$*-graph.*

- *For all* $n \in N$ *we define the formula* $\varphi(n)$ *as:*

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

- *For all edges* $e = \langle n_1, \langle c, p \rangle, n_2 \rangle \in E$*, the formula* $\varphi(e)$ *is defined as:*

$$\varphi(e) = \begin{cases} p & \text{if } \ell_{\mathcal{C}}(e) = \ell_{\mathcal{C}}(n_2) \\ \overline{p} & \text{otherwise} \end{cases}$$

Example 6.4 *Let us consider the fourth* $\mathbb{W}$*-query (from left to right) of Figure* 1 *whose semantics is that of finding teachers who teach at least a course, but not the Database one.*

If n_1, n_2, n_3 *are the three nodes, from top to bottom, it holds that* $\varphi(n_1) =$ Teacher*,* $\varphi(n_2) =$ Course*, and* $\varphi(n_3) =$ Databases*. Moreover, if* e_1 *is the edge from* n_1 *to* n_2 *and* e_2 *the edge from* n_2 *to* n_3*, it holds that:* $\varphi(e_1) =$ teaches*,* $\varphi(e_2) =$ cname*.*

6.3 Acyclic Queries

Definition 6.5 *Let* $G = \langle N, E, \ell \rangle$ *be an acyclic* $\mathbb{W}$*-graph, and* $\nu \in N$*. Then the formula* $\Psi_\nu(G)$*, depending on both* G *and* ν *is defined recursively as follows (cf. Fig.* 5*):*

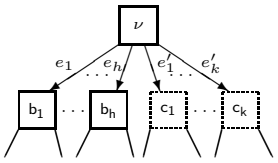


Figure 5: Graph for computing $\Psi_\nu(G)$.

- *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. $\ell_{\mathcal{C}}(c_i) = dashed$.*
- *For $i = 1, \ldots, h$ and $j = 1, \ldots, k$ let e_i be the edge which links ν to b_i and e'_j the one which links ν to c_j and let $\Psi_{b_i}(G)$ and $\Psi_{c_j}(G)$ be the formula computed in the nodes b_i and c_j.*

If $\ell_{\mathcal{C}}(\nu) = solid$, then

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

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

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

Example 6.6 *Let us consider again the fourth $\mathbb{W}$-query (from left to right) of Figure 1, that we refer as G, and let n_1, n_2, n_3, e_1, e_2 be its nodes and edges as in Example 6.4. Then:*

- $\Psi_{n_3}(G) = \neg\varphi(n_3) = \neg\mathsf{Database}$
- $\Psi_{n_2}(G) = \varphi(n_2) \wedge \mathsf{AX}_{\varphi(e_2)}(\Psi_{n_3}(G)) = \mathsf{Course} \wedge \mathsf{AX}_{\mathsf{cname}}\neg\mathsf{Database}$
- $\Psi_{n_1}(G) = \varphi(n_1) \wedge \mathsf{EX}_{\varphi(e_1)}(\Psi_{n_2}(G)) = \mathsf{Teacher} \wedge \mathsf{EX}_{\mathsf{teaches}}(\mathsf{Course} \wedge \mathsf{AX}_{\mathsf{cname}}\neg\mathsf{Database})$.

The recursive definition of the formula and the acyclicity of G ensures that the formula built is a *CTL* formula (cf. Def. 2.2).

6.3.1 Acyclic Accessible $\mathbb{W}$-graphs

Definition 6.7 *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 $\Psi_\nu(G)$.*

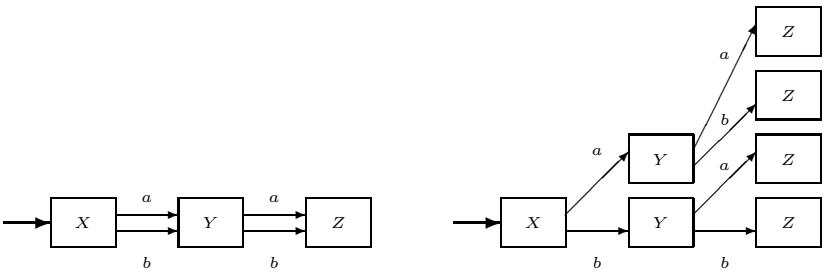


Figure 6: The two graphs produce the formula $X \wedge \mathsf{EX}_a(Y \wedge \mathsf{EX}_a(Z) \wedge \mathsf{EX}_b(Z)) \wedge \mathsf{EX}_b(Y \wedge \mathsf{EX}_a(Z) \wedge \mathsf{EX}_b(Z))$

Remark 6.8 *Observe that:*

1. *Each node and edge of G is used to build the formula.*

2. *The size of the formula $\Psi_\nu(G)$ can grow exponentially w.r.t. $|G|$, since the construction of the formula could involve the unfolding of a DAG. For instance, the two graphs in Fig. 6 produces the same formula, which has the structure of the rightmost graph. The problem is that in the explicit representation of a formula, subformulas must be repeated many times. However, model-checking procedures deal properly on compact representations of formulas. This allows to store the formula keeping the memory allocation (and the execution time) linear w.r.t. $|G|$.*

6.3.2 Acyclic $\mathbb{W}$-solid graphs

The condition on the accessibility of all nodes of G for ν can be weakened, in absence of cycles. Consider, for instance, the two goals of Fig. 7. If works is the inverse relation of gives

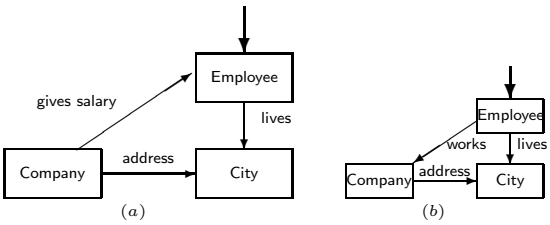


Figure 7: Two equivalent acyclic queries

salary (i.e., gives salary^{-1}), then one expects the same results from query (a) and query (b). Thus, the idea is to swap the direction of some edges, replacing the labeling relation with its inverse.[2] This procedure can be automatized:

Procedure 6.9 *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 s.t. $\hat{E} = \{\{m, n\} : \langle m, \ell, n\rangle \in E\}$.*
2. *Identify each connected component of $\hat{G}$ by one of its nodes. Use ν for its connected component. Let $\mu_1, \ldots, \mu_h$ be the other nodes chosen.*
3. *Execute a breadth-first visit of $\hat{G}$ starting from ν, then from $\mu_1, \ldots, \mu_h$.*
4. *Consider the list of nodes $\nu = n_0, n_1, \ldots, n_k$ ordered by the above visit.*
5. *Build the $\mathbb{W}$-graph $\vec{G} = \langle N, \vec{E}, \ell\rangle$ from G by swapping the edges that are not consistent with the ordering of the previous point:*

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

This procedure 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 $(\nu, \mu_1, \ldots, \mu_h)$ since they have been reached by a visit. Procedure 6.9 can be implemented in time $O(|N| + |E|)$. Thus, for each node $\nu, \mu_1, \ldots, \mu_h$ we can compute the formulae $\Psi_\nu(\vec{G}), \Psi_{\mu_1}(\vec{G}), \ldots, \Psi_{\mu_h}(\vec{G})$.
We recall here the semantics of the formula $\mathsf{EF}_a(\phi)$ that we use below (see also Sect. 2):

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

Given a state s, $s \models \mathsf{EF}_a(\phi)$ 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 6.10 *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. Let $\mathcal{A}ct_G$ is as in Def. 5.1; the formula associated with Q is:*

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

Observe that Procedure 6.9 terminates even if G admits cycles (save self-loops). However, in these cases, the semantics of the query is lost. Cyclic queries requires different modal operators (see Subsection 6.6.4).

For this kind of graphs we can state the equivalence result w.r.t. the bisimulation-based semantics of Section 4. In the following theorem we identify with $m_1, m_2, \ldots$ the nodes of a $\mathbb{W}$-instance I and with $m'_1, m'_2, \ldots$ the corresponding nodes in the KTS $\mathcal{K}_I$.

Theorem 6.11 *Let I be a $\mathbb{W}$-instance and $Q = \langle G, \nu \rangle$ be an accessible, acyclic solid $\mathbb{W}$-query. Then, a graph $C \sqsubseteq I$ is bisimilar to G with nodes $m_1, \ldots, m_z$ in relation with ν if and oly if the corresponding states $m'_1, \ldots, m'_z$ in $\mathcal{K}_I$ are s.t. $m'_i \models \Psi_\nu(G)$.*

Proof. We prove the property by induction on *depth*(ν), the maximum length of a directed path from ν to a leaf.

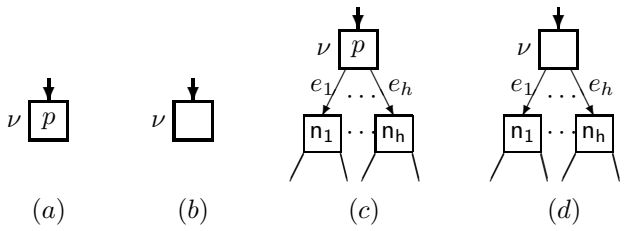


Figure 8: Queries of *depth* 0 and greater than 0

If *depth*(ν) = 0, then the query $Q = \langle G, \nu \rangle$ is a graph of the type (a) or of the type (b) in Figure 8.

In the case (a), G is bisimilar to all instance nodes m such that $\ell_{\mathcal{L}}(m) = p$. Assume they are $m_1, \ldots, m_z$. The formula $\Psi_\nu(G) = \varphi(\nu) = p$ holds in all the states m' of $\mathcal{K}_I$ such that $p \in \mathcal{I}(n)$. By construction of $\mathcal{K}_I$, those states are exactly $m'_1, \ldots, m'_z$.

The case (b) is trivial: by definition of bisimulation regarding dummy nodes, all the nodes of the instance I are in relation to ν. Similarly, The formula $\Psi_\nu(G) = \mathsf{true}$ holds in all the states of $\mathcal{K}_I$.

Let *depth*(ν) > 0. The query $Q = \langle G, \nu \rangle$ is a graph of the type (c) or of the type (d) in Figure 8. Let us consider the case (c). By construction, $\Psi_\nu(G) = p \wedge \bigwedge_{i=1\ldots h} \mathsf{EX}_{\varphi(e_i)}(\Psi_{n_i}(G))$.
($\rightarrow$) Assume $C \sqsubseteq I$ and $C \overset{b}{\sim} G$ and $m_1\ b\ \nu, \ldots, m_z\ b\ \nu$. By definition of bisimulation, it holds that $\ell_{\mathcal{L}}(m_i) = p$ for all $i \in \{1, \ldots, z\}$. Thus, by construction of $\mathcal{K}_I$, p holds in all the states m'_i. By definition of bisimulation, for any edge $\langle \nu, e_i, n_i \rangle$ and for each m_i there is in C one edge outgoing from m_i with label e_i and the node ξ_i reached is s.t. $\xi_i \overset{b}{\sim} n_i$. Thus, from the definition of bisimulation and from the acyclicity of the query, it follows that there is a subgraph of C rooted at ξ_i bisimilar to the subquery of Q rooted at n_i.

By inductive hypothesis, the nodes ξ'_is are such that $\xi'_i \models \Psi_{n_i}(G)$. By the semantics of the modal operator EX, each node m'_i is such that the formula $p \wedge \bigwedge_{i=1\ldots h} \mathsf{EX}_{\varphi(e_i)}(\Psi_{n_i}(G))$ holds.

($\leftarrow$) We prove the property for each state m' that satisfies the modal formula. Assume a state m' in $\mathcal{K}_I$ is such that $m' \models p \wedge \bigwedge_{i=1\ldots h} \mathsf{EX}_{\varphi(e_i)}(\Psi_{n_i}(G))$. By construction of $\mathcal{K}_I$, this means that p is the label of the corresponding node m. By the semantics of the modal operator EX there are states $\xi'_1, \ldots, \xi'_k$ such that they are related to m' by relations (i.e., labeled edges) $e_1, \ldots, e_k$, respectively, and such that $\xi'_i \models \Psi_{n_i}(G)$.

By inductive hypothesis, there are graphs $C_1, \ldots, C_k$ (all of them subgraphs of I) rooted at nodes $\xi_1, \ldots, \xi_k$ such that there are bisimulations $b_1, \ldots, b_k$ with the subgraph of Q rooted at $n_1, \ldots, n_h$, respectively.

Thus, we can define the relation b as follows: $b = b_1 \cup \cdots \cup b_k \cup \{\langle m, \nu \rangle\}$. We prove that b is a bisimulation between a subgraph C of G m and the query Q. Properties (1), (2), and (3) of Def. 4.1 holds trivially for all the graphs but the toot nodes since $b_1, \ldots, b_h$ are bisimulations. Let us see the situations for the roots. Since $m\ b\ \nu$ by definition, property (1) holds. Since the nodes m and ν are both labeled by p, property (2) holds. The presence of relations e_i betweeen m' and ξ'_i, by construction of $\mathcal{K}_I$, ensures that there is in I an edge labeled e_i from m to ξ_i, for all $i = 1, \ldots, k$. This ensures that property (3) holds.

Now, assume that nodes $m'_1, \ldots, m'_k$ are all s.t. $m'_i \models \Psi_\nu(G)$. We have just proved that there are subgraphs $C_1, \ldots, C_k$ of I rooted at $m_1, \ldots, m_k$ and bisimilar to the query Q with bisimulations, say, $b_1, \ldots, b_k$ and such that $m_1 \overset{\sim}{b_1} \nu, \ldots, m_k \overset{\sim}{b_k} \nu$. It is immediate to check that the graph obtained by union of $C_1, \ldots, C_k$ is bisimilar to Q using the bisimulation $b_1 \cup \cdots \cup b_k$ (a simple property of bisimulation).

If Q is a graph of type (d) the proof above can be repeated replacing p by $\bot$ for the bisimulation side and by true for the CTL side. □

6.3.3 Hidden accessible $\mathbb{W}$-graphs

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

Definition 6.12 *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.*
2. *Apply Procedure* 6.9 *to G_s. Let $\nu, \mu_1, \ldots, \mu_h$ be the root nodes chosen.*
3. *Swap in G the same edges as in Step* 2 *of Procedure* 6.9*, obtaining the graph $\vec{G}$.*
4. *If $\vec{G}$ is acyclic, then compute the formulae $\Psi_\nu(\vec{G}), \Psi_{\mu_1}(\vec{G}), \ldots, \Psi_{\mu_h}(\vec{G})$*
5. *If all the nodes of G have been visited during the last phase, we call Q an* Hidden accessible $\mathbb{W}$-graph *and the formula associated with Q is: $\Psi_\nu(\vec{G}) \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_1}(\vec{G})) \wedge \cdots \wedge \mathsf{EF}_{\mathcal{A}ct_G}(\Psi_{\mu_h}(\vec{G}))$.*

The semantics of a query may change when the direction of dashed edges is modified. Consider, for example, the query (a) in Fig. 10, and its associated formula:

$$\mathsf{Teacher} \wedge \mathsf{AX}_{\overline{\mathsf{teaches}}}(\neg\mathsf{Course}) \wedge \mathsf{EF}_{\mathcal{A}ct_{\mathsf{G}(\mathsf{a})}}(\mathsf{Student} \wedge \mathsf{AX}_{\overline{\mathsf{attends}}}(\neg\mathsf{Course}))$$

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

Procedure 6.9 should replace the edge labeled attends in the query (a) with an edge labeled $\mathsf{attends}^{-1}$, as depicted in the query (b) of Fig. 10. The formula associated with this query is:

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

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

6.4 Cyclic queries

In this section we assign a propositional temporal logic semantics to some $\mathbb{W}$-queries with cycles. We then discuss some possible extensions of our proposal by using other logics.

To start, let us consider queries (b) and (d) of Figure 9. The meaning of (d) is known: look for all the functions that call a function. Its CTL semantics is

$$\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{calls}}\ \mathsf{Function}$$

Query (b) requires a stronger constraint. The second function must call a function itself. The bisimulation-based semantics of [16] consider them as different. We need to find a temporal logic semantics for cyclic queries that preserves their intuitive meaning.

Let us consider queries composed by nodes with a unique label ℓ_n and edges all labeled ℓ_e, that we call *simple cycle queries*. We can assign a *CTL* formula to them, by using the Generally operator, used as $\mathsf{EG}_a(\phi)$, whose semantics is (see also Sect. 2):

> Given a state s, $s \models \mathsf{EG}_a(\phi)$ iff there is an infinite 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 $\ell_j \in a)$.

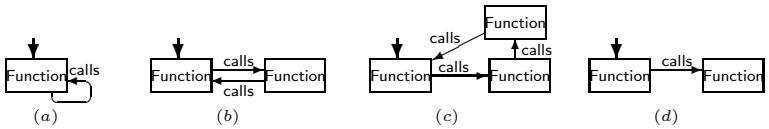


Figure 9: Three bisimilar simple cyclic queries and an acyclic one

Consider for example, the cyclic query (a) in Fig. 9, it requires to collect *all the functions which are recursive*. Its behavior could be described by:

$$\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{calls}}(\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{calls}}(\mathsf{Function} \wedge \mathsf{EX}_{\mathsf{calls}}(\cdots)))$$

that can be mimicked exactly by $\mathsf{EG}_{\mathsf{calls}}\mathsf{Function}$. Queries (a), (b), and (c) can be considered equivalent, since all their models satisfy the formula $\mathsf{EG}_{\mathsf{calls}}\mathsf{Function}$. This fact has a graph counterpart: their pointed graphs are bisimulation equivalent and (a) is the minimum graph bisimulation equivalent (there is a unique such a minimum graph, called bisimulation contraction [4]). It is easy to extend the proof of Theorem 6.11 to this kind of queries.

More complex cyclic queries could be translated by using the richer temporal logic *CTL**, which extends *CTL* by allowing basic temporal operators where the path quantifier (A or E) is followed by an arbitrary linear-time formula, allowing Boolean combinations and nesting, over F, G, X, and U [22].[3] Consider the query (c) in Fig. 10. It requires to collect *all the classes which call a function defined inside themselves*. The *CTL** formula is:

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

Observe that the modal operator X is used without any path quantifier. This is allowed in *CTL** but not in *CTL*. In particular, a path $\pi = \langle \pi_0, \ell_0, \pi_1, \ell_1 \ldots \rangle$ satisfies the formula $\mathsf{X}_a\psi$ when $\ell_0 \in a$ and ψ holds starting at the second state π_1 on the path.

The first part of the formula $\Psi_{(c)}$ above is aimed at identifying cycles of length greater than or equal to 2, the Generally operator imposes to retrieve only cyclic paths where nodes labeled Class alternate with nodes labeled Function.

In general, given a $\mathbb{W}$-query $\langle G, x_1 \rangle$, where G is a simple cycle composed by n different nodes $x_1, \ldots, x_n$, then the corresponding *CTL** formula is

$$\begin{aligned}\Psi_{x_1}(G) = \; & \varphi(x_1) \wedge \mathsf{EG}_{\{\ell_1,\ldots,\ell_n\}}\big((\varphi(x_1) \vee \cdots \vee \varphi(x_n)) \wedge \\ & (x_1 \rightarrow \mathsf{X}_{\ell_1}\varphi(x_2)) \wedge \cdots \wedge (x_n \rightarrow \mathsf{X}_{\ell_n}\varphi(x_1))\big)\end{aligned}$$

It is possible to prove (the proof similar to the one for Theorem 6.11) that a subset of the logic *CTL** can exactly translate, w.r.t the bisimulation based semantics of the $\mathbb{W}$-query language, these kinds of cyclic queries.

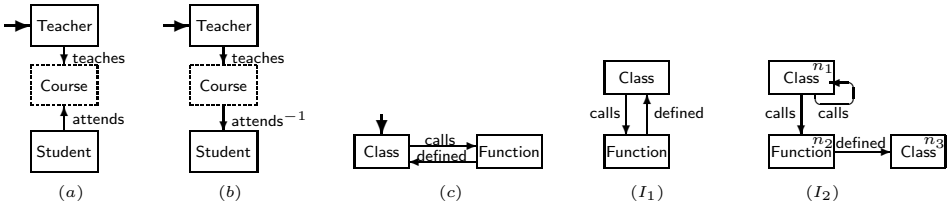


Figure 10: $\mathbb{W}$-queries and $\mathbb{W}$-instances

We remark here that with the *CTL* logic it is only possible to approximate the translation of simple cyclic queries (with more than one node). In fact, a *CTL* formula for the query (c) in Fig. 10 could be:

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

[3] Precisely, given a set $\mathcal{A}ct$ of actions and a set Π of atomic propositions, the set *Lit* of *literals* is defined as $Lit = \Pi \cup \{\neg q \mid q \in \Pi\} \cup \{\mathsf{true}, \mathsf{false}\}$. *State formulae* ϕ and *Path formulae* ψ are inductively defined by the following grammar, where $p \in Lit$ and $a \subseteq \mathcal{A}ct$:

$$\begin{array}{lll}\text{state formulae:} & \phi & ::= \; p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{A}\psi \mid \mathsf{E}\psi \\ \text{path formulae:} & \psi & ::= \; \phi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathsf{G}_a\psi \mid \mathsf{F}_a\psi \mid \mathsf{X}_a\psi \mid \mathsf{U}_a(\psi,\psi)\end{array}$$

The node Class of instance (I_1) in Fig. 10 satisfies both the formulas $\Psi_{(c)}$ and $\Psi'_{(c)}$. Instead the node n_1 of instance (I_2) in Fig. 10 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 could accept this loosing of precision, and assign a formula to a (pointed) cycle.

Definition 6.13 *The formula $\Psi_{\nu_1}(G)$ associated to a solid cyclic graph*

$$G = \langle \{\nu_1, \ldots, \nu_n\}, \{\langle \nu_1, \ell_1, \nu_2\rangle, \langle \nu_2, \ell_2, \nu_3\rangle, \ldots, \langle \nu_n, \ell_n, \nu_1\rangle\}, \ell, \nu_1\rangle,$$

is defined as:

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

where $\varphi(\nu_1)$ is as in Def. 6.3*, and ψ_C is the CTL formula $\Psi_{\nu_1}(C)$ associated to the DAG*

$$C = \langle \{\nu_1, \ldots, \nu_{n+1}\}, \{\langle \nu_1, \ell_1, \nu_2\rangle, \langle \nu_2, \ell_2, \nu_3\rangle, \ldots, \langle \nu_n, \ell_n, \nu_{n+1}\rangle\}\rangle,$$

rooted at ν_1 and where ν_{n+1} is a "copy" of ν_1, i.e., a new node s.t. $\ell(\nu_{n+1}) = \ell(\nu_1)$.

To sum up, consider the $\mathbb{W}$-query (a) in Fig. 11. In order to find its *CTL* encoding we apply the rewriting procedure to swap the in edge, obtaining the $\mathbb{W}$-query (b) and the formula:

$$\psi_\nu = \mathsf{Class} \wedge \mathsf{EX}_{\overline{\mathsf{isa}}}(\mathsf{Class} \wedge \mathsf{EX}_{\mathsf{name}}\mathsf{Math}) \wedge \mathsf{EX}_{\mathsf{in}^{-1}}(\mathsf{Procedure} \wedge \mathsf{EG}_{\mathsf{calls}}\mathsf{Function})$$

Note that the *CTL* formula associated to the simple cyclic query rooted at n is $\mathsf{EG}_{\mathsf{calls}}\mathsf{Function}$.

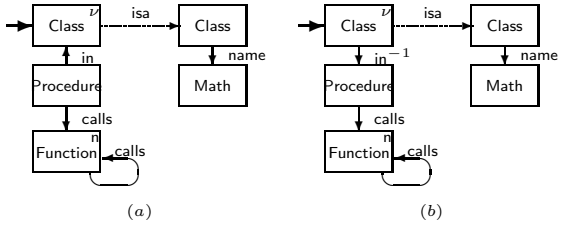


Figure 11: Two $\mathbb{W}$-queries

Remark 6.14 *CTL and CTL* distinguish each state of a Kripke structure by means of the set of properties that hold in the state. Thus, a limit of these logics is the impossibility to refer to a given node, as needed, for instance, for a* Join *operation. Recently introduced Hybrid Logics [6] could overcome this limitation. Hybrid Logics allow to make explicit reference to states. A deep analysis of the possibilities of these new logics for implementing cyclic queries and* Join *conditions are under study.*

Moreover, other families of queries of SQL-like languages cannot be translated into CTL, due to the fact that a model-checker cannot elaborate previously retrieved information and thus, cannot deal with typical aggregation functions of SQL.

7 Complexity issues

The following theorem proves the polinomiality of the data retrieval problem for the classes of queries described.

Theorem 7.1 *Let $\langle G, \nu \rangle$ be a $\mathbb{W}$-query in one of the forms described in Section* 6.6.2. *Querying a $\mathbb{W}$-instance I with G can be done in linear running time on $|\mathcal{K}_I| \cdot |\Psi_\nu(G)|$.*

Proof. The queries of the form in the hypothesis of the theorem can be translated into CTL formulas. Then, the thesis is inherited by the classical result in [14]. □

Corollary 7.2 *Querying a $\mathbb{W}$-instance I with a $\mathbb{W}$-query G in one of the forms described in Section* 6.6.2 *if in G we do not ask for paths with negated actions, can be done in time linear on $|I| \cdot |G|$.*

Proof. As shown in Section 5 $\mathcal{K}_I$ can be computed from I using a quadratic time. Moreover, a quadratic space complexity is introduced as negative relations are computed and stored. However, if in the query we do not ask for conditions AX_a, EX_a, where a is a negated action, we do not need to store negated edges. In this way we have exactly twice the number of initial edges and the computation of $\mathcal{K}_I$ and its size depend linearly on $|I|$.

As far as the size of the formula is concerned, as discussed in Remark 6.8, 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 and thus it can be used when processing it.

The result then follows by Theorem 7.1. □

As shown in Sect. 6.6.4 for cyclic queries, it seems to be impossible to map all the queries in *CTL* formulae. This fact can be justified algorithmically. The semantics of acyclic $\mathbb{W}$-queries without negation can be proved to be equivalent to that of G-Log queries without negation (Theorem 6.11). If we extended this equivalence result to cyclic queries (even without negation), we would provide an implementation of a subset of G-log in which data retrieval is equivalent to the subgraph bisimulation problem, proved to be NP complete in [20]. This would be in contradiction with Theorem 7.1.

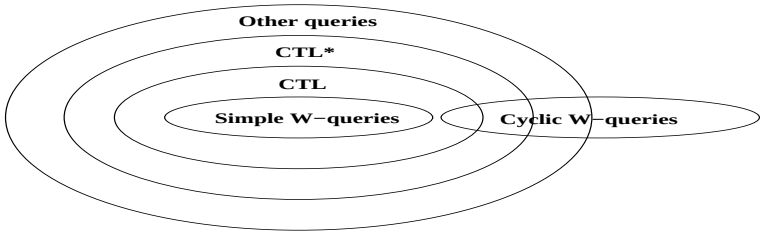


Figure 12: Complexity of $\mathbb{W}$-queries with respect to *CTL* and *CTL**

8 Implementation of the method

We have effectively tested the data retrieval technique presented in the previous sections by using the model-checker NuSMV [13], which is designed to check the satisfaction of specifications expressed in *CTL* (or *LTL*) on Kripke Structures. To this aim, KTS related to $\mathbb{W}$-instances have been rewritten into Kripke Structures, where edges are not labeled (see Rem. 2.5). Rewriting is done as follows.

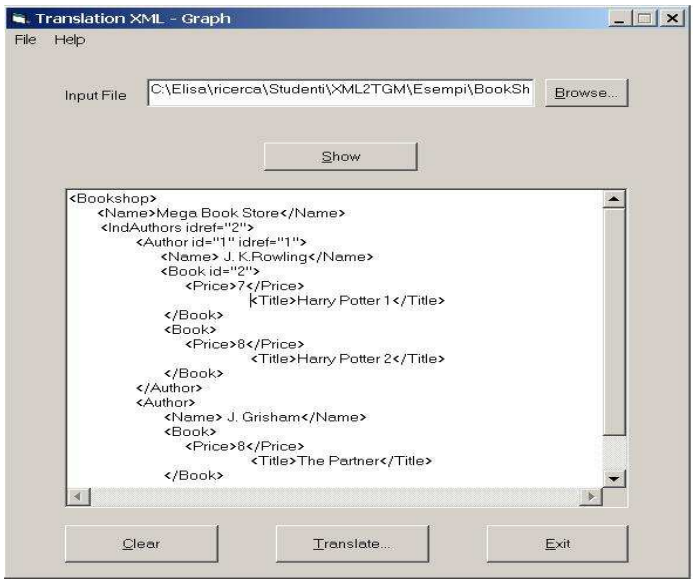


Figure 13: A sample XML document

Procedure 8.1 *Given a KTS, 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.*

We have evaluated the proposed approach by performing a set of experiments either on real and synthesized XML documents. We have tested some queries (see Fig. 16) on the XML document of Fig. 13, which is a simple example containing all the features we can handle with our approach, such as ID/IDREF formalism used here to obtain a cyclic graph. We have developed a tool to convert XML documents into graphs. Attributes and leaf elements are represented by oval symbols; these nodes have a label composed by a name (the attribute name or the element tag) and a content (the value of the attribute or the textual content of leaf elements). The document obtained is represented in Fig. 14 and it is then coded into the KS of Fig. 15. The set of states of the KS is chosen by declaring the state variable `s` to assume values `n0`, ..., `n15` (the instance's node identifiers). The transition relation of the KS is expressed by assigning, for each value of the variable `s`, the list of nodes that can be reached from it using one edge. The variables `l` and `c` are introduced to define the label (tag and content in the XML context) of each state, identified by the value of the variable `s`.

All the edges in the considered XML document have the same label sub-element of. Thus, labeled nodes are not needed to be introduced in the KS (cf. Procedure 8.1). For queries that require a backward traversing of the graph and/or queries with negative requirements or universally quantified conditions (e.g., R_4 of Fig. 4) we should consider edges labeled sub-element of^{-1} and negated edges, as explained in Section 5. In this example we have omitted these edges and added self-loops to leaves (cf. Remark 5.3).

Let us explain $\mathbb{W}$-queries of Fig. 16: (Q_1) requires all the Bookshop elements which do not have an Author sub-element (but an Author element must exist in the document) and (Q_2) all the Bookshop elements which do not have any Book sub-element. Query (Q_3) is not a tree: it requires to collect all the elements Author which have an IDREF to themselves and have a Name as sub-element. Query (Q_4) contains only the cyclic condition of (Q_3).

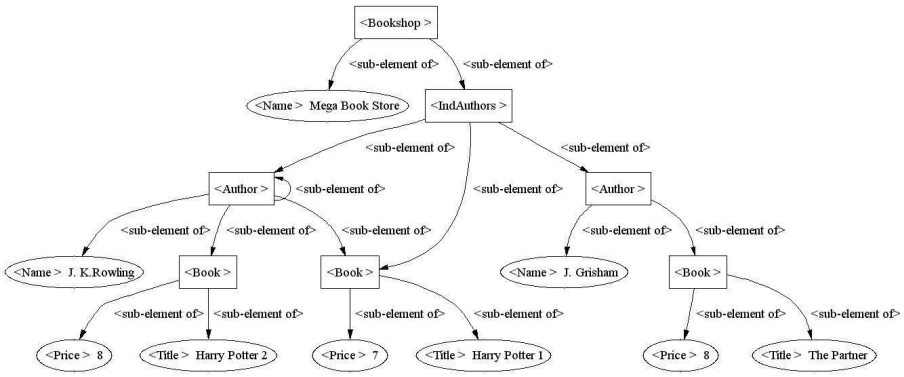


Figure 14: The graph-based representation of the XML document in Fig. 13

The translation of the four queries in the NuSMV format is in Fig. 17.

NuSMV is a procedure that decides whether a given KS M is a model of a *CTL* formula Ψ, i.e., whether M satisfies Ψ, and returns as main output only `true` or `false`. In our application we are interested in finding out the set of states in M that satisfy Ψ. To this purpose we have extended the NuSMV code in order to solve the global model-checking problem, i.e., to print all the states of a given KS that satisfy Ψ. For the above examples, we have obtained the answers in Fig. 17.

As a real XML document we have used the ACM SIGMOD Web site produced within the Araneus project [42]. First, we have translated it in a $\mathbb{W}$-graph (actually, a tree) by using the Java DOM API of W3C. We have noted that the only tag with some irregularities is the `AUTHORS` tag: for some articles `AUTHORS` is an element with a textual content, for the others, it contains some sub-elements with tag `AUTHOR`, each of them reporting the name of an author. When constructing the $\mathbb{W}$-tree we have not considered the `AUTHOR` elements and thus, we have reported the name of the paper authors as textual content of the respective `AUTHORS` element. The resulting XML document is quite structured and thus, the NuSMV specification of the corresponding KTS is rather compact.

We report, in Fig. 18, a fragment of the produced file. Three queries that we have tested on the KS are the following:

```
1.    label=SigmodRecord & EF(label=articlesTuple & !EX(label=initpage & content=5))
2.    label=SigmodRecord & EF(label=articlesTuple & EX(label=initpage & content=4))
3.    label=SigmodRecord & AX(label != articlesTuple)
```

On this real dataset we have performed some more experiments by varying a) the instance size (and thus the XML file size) and b) the query size, and applying the query on an instance composed by 601 edges and 336 nodes. The results we have obtained on a 33MHz Pentium II system with 512KB RAM, running RedHat Linux 2.4 are shown in Fig. 19 and 20, respectively.

This test confirms the possibility to solve queries on real XML documents by using model-checking based polynomial time methods. However, in order to really use model-checkers we should deeply study how to store KS structures (possibly encoded as OBDDs as needed by model checkers) into secondary memory and how to deal with regularities of XML documents (expressed into DTD) in order to reduce the size of the related KS.

```
MODULE main
VAR
s   : {n0, ..., n15};
lab : {Bookshop, Name, IndAuthors, Author, Book, Price, Title, MegaBookStore;
       JKRowling, 7, 8, HarryPotter1, HarryPotter2, JGrisham, ThePartner, nil};
ASSIGN
   init(s) := {n0};
   next(s) := case
      s = n0 : {n1, n2}; s = n1 : n1; s = n2 : {n3, n5, n11};
      s = n3 : {n4, n3, n5, n8}; s = n4 : n4; s = n5 : n6, n7;
      s = n6 : n6; s = n7 : n7; s = n8 : n9, n10;
      s = n9 : n9; s = n10 : n10; s = n11 : n12, n13;
      s = n12 : n12; s = n13 : n14, n15; s = n14 : n14; s = n15 : n15;
   esac;
DEFINE
   l := case
      s = n0 : Bookshop; s = n1 : Name; s = n2 : IndAuthors; s = n3 : Author;
      s = n4 : Name; s = n5 : Book; s = n6 : Price; s = n7 : Title;
      s = n8 : Book; s = n9 : Price; s = n10 : Title; s = n11 : Author;
      s = n12 : Name; s = n13 : Book; s = n14 : Price; s = n15 : Title;
   c := case
      s = n1 : MegaBookStore; s = n4 : JKRowling; s = n6 : 7;
      s = n7 : HarryPotter1; s = n9 : 8; s = n10 : HarryPotter2; s = n12 : JGrisham;
      s = n14 : 8; s = n15 : ThePartner; 1 : nil;
   esac;
```

Figure 15: NuSMV format of a $\mathbb{W}$-instance

9 Applications to existing languages

In this section we apply the model-checking based data retrieval approach to the query languages for semistructured data XPath, UnQL, GraphLog, and G-Log.

9.1 XPath

Our idea of building a bridge from Semistructured Databases to Model Checking, first published in [21], is used in several recent papers concerning the translation of subclasses of the language XPath [41] to CTL formulas. XPath is a language for selecting nodes from XML document trees. A concise and clear semantics of the language is given in [27]. We give here only the idea of the translation, (which basically follows the lines described in

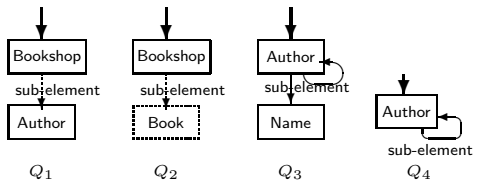


Figure 16: Sample Queries

```
SPEC
   l = Bookshop & ¬EX(l = Author) & EF(l = Author)
The states which satisfy the formula are:
   State0 :  l = Bookshop  content = nil
SPEC
   l = Bookshop & AX(¬(l = Book))
The states which satisfy the formula are:
   State0 :  l = Bookshop  content = nil
SPEC
   EG(l = Author) & EX(l = Name)
The states which satisfy the formula are:
   State3 :  l = Author  content = nil
SPEC
   l = Author & EG(l = Author)
The states which satisfy the formula are:
   State3 :  l = Author  content = nil
```

Figure 17: Translation of Queries in Fig 16 and Answers

this paper) leaving details to recent papers by the group of Gottlob [24, 26, 28] and by the group of Marx and de Rijke [30, 5].

In [27] it is identified a core of the language, called Core XPath, based on the notion of *location path* that is defined by the context free grammar in Figure 21. tag denotes labels addressing data and $*$ is a symbols that matches with all tags. XPath queries are based on locpaths (actually, XPath queries are locpaths with a further option for Boolean expressions Bexpr—cf. [28]). Extending the proposal in [25] to the definition of Core XPath of [27] (see Fig. 21), we show how each location path can be associated to a *CTL* formula to be verified on a KTS obtained from a XML document. XML documents can be considered as directed graphs where edges are not labeled. For consistency with our notation, we assume that all edges are labeled by the unique label $\triangleright$. In the construction of the KTS associated to a XML document (cf. Sect. 5) we add all inverse edges, labeled $\triangleleft$. In Figure 22 we define the function τ mapping location paths to *CTL* formulas. With $\varphi_1 \circ \varphi_2$ we denote a formula obtained by composition of the two formulas φ_1 and φ_2. Precisely, into each innermost quantifier free subformula of φ_1 we add $\wedge\varphi_2$.

Example 9.1 *1. Let us consider the query* $G = \mathsf{child} :: a/\mathsf{descendant} :: b$ *that means: collect all the nodes that have a children node labeled* a *that have a descendant node labeled* b*. Then,* $\tau(\mathsf{child} :: a) = \mathsf{EX}_{\triangleright}(a)$*,* $\tau(\mathsf{descendant} :: b) = \mathsf{EF}_{\triangleright}(b)$*.* $\tau(G)$ *is obtained by composition* $\circ$ *as* $\mathsf{EX}_{\ell}(a \wedge \mathsf{EF}_{\ell} b)$*.*

2. Let $G' = /\mathsf{child} :: a[\mathsf{descendant} :: b|\mathsf{parent} :: \mathsf{root}]/\mathsf{descendant} :: c$*. Then*

$$\tau(G') = \mathsf{root} \wedge \mathsf{EX}_{\triangleright}(a \wedge (\mathsf{EF}_{\triangleright}(b \underbrace{\wedge \mathsf{EF}_{\triangleright}(c)}_{1}) \vee \mathsf{EX}_{\triangleleft}(\mathsf{root} \underbrace{\wedge \mathsf{EF}_{\triangleright}(c)}_{2})))$$

Observe that in this case the highlighted subformula is copied twice for realizing the composition.

Example 9.2 $\mathbb{W}$*-queries* Q_1 *and* Q_2 *of Fig.* 16 *can be translated into Core XPath as follows.* $Q_1 = \mathsf{self} :: \mathsf{Bookshop}[$ *not* $\mathsf{child} :: \mathsf{Author}$ *and* $\mathsf{ancestor} :: \mathsf{root}/\mathsf{descendant} :: \mathsf{Author}]$ *and* $\tau(Q_1) = \mathsf{Bookshop} \wedge \neg\mathsf{EX}_{\triangleright}\mathsf{Author} \wedge \mathsf{EF}_{\triangleleft}(\mathsf{root} \wedge \mathsf{EF}_{\triangleright}\mathsf{Author})$*.* $Q_2 = \mathsf{self} ::$

```
MODULE main
VAR
    node : 1..1326;
    labels : {SigmodRecord,issues,issuesTuple,volume,number, ..., nil};
ASSIGN
     init(node):=1;
     next(node):= case
        node=1 | node=2     : node+1;
        node=3              : {node+1,node+2,node+3};
        node=4 | node=5     : node;
        node=6              : {7,12,17, ... ,1317,1322};
        (node - 7)mod 5 =0  : {node+1,node+2,node+3,node+4};
        (node - 8)mod 5 =0  : node;
        (node - 9)mod 5 =0  : node;
        (node -10)mod 5 =0  : node;
        (node -11)mod 5 =0  : node;
     esac;
DEFINE
    label:= case
        node=1 : SigmodRecord; node=2 : issues; node=3 : issuesTuple;
        node=4 : volume; node=5 : number; node=6 : articles;
        (node - 7)mod 5=0       : articlesTuple;
        (node - 8)mod 5=0       : title;
        (node - 9)mod 5=0       : initpage;
        (node -10)mod 5=0       : endpage;
        (node -11)mod 5=0       : authors;
    esac;
    content:=  case
            node=4          : 1;
            node=5          : 2;
            node=8          : TitleContentsSketchesPublicationsInterestingHappenings;
            ...
            node=76         : DanielGBobrowWilliamRSutherland;
            1               : nil;
         esac;
```

Figure 18: The file SigmodRecord.smv

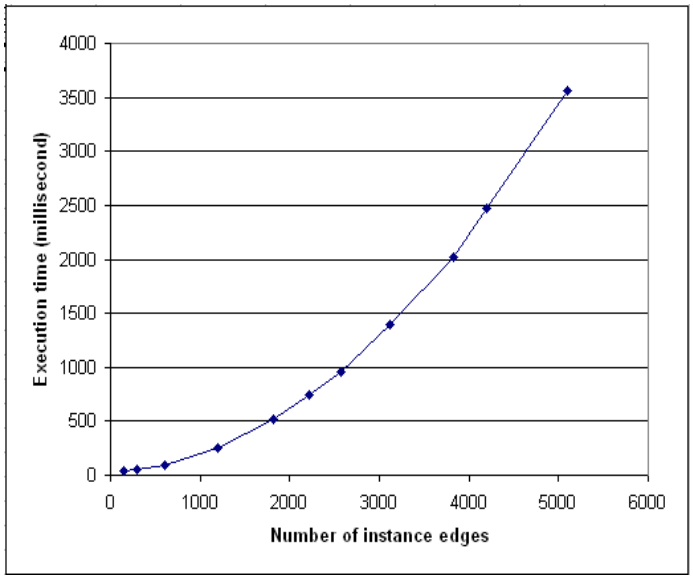


Figure 19: Experiment results: instance size and execution time

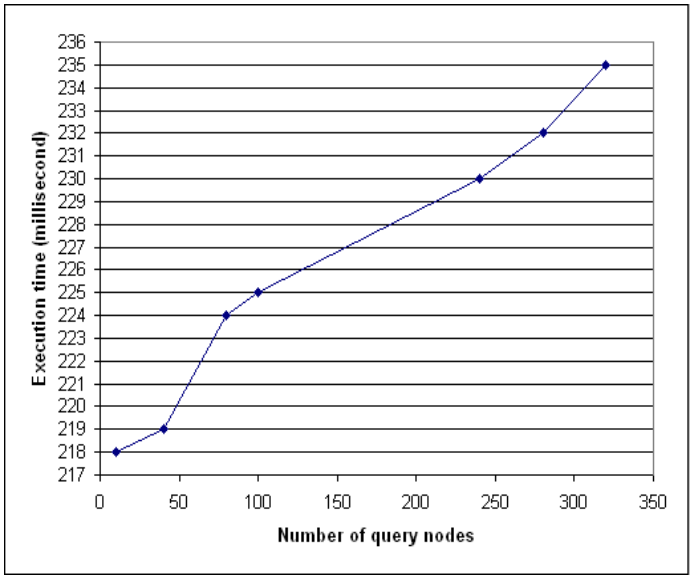


Figure 20: Experiment results: query size and execution time

$$
\begin{array}{rcl}
\langle \mathsf{locpath} \rangle & \rightarrow & \text{“/”}\langle \mathsf{locpath} \rangle \mid \langle \mathsf{locpath} \rangle \text{“/”}\langle \mathsf{locpath} \rangle \mid \\
 & & \langle \mathsf{locpath} \rangle \text{“|”}\langle \mathsf{locpath} \rangle \mid \langle \mathsf{locstep} \rangle \\
\langle \mathsf{locstep} \rangle & \rightarrow & \langle \mathsf{axis} \rangle \text{“ :: ”}\langle \mathsf{nodetest} \rangle \langle \mathsf{Blist} \rangle \\
\langle \mathsf{nodetest} \rangle & \rightarrow & \mathsf{tag} \mid \text{“}*\text{”} \\
\langle \mathsf{Blist} \rangle & \rightarrow & \varepsilon \mid \text{“[”}\mathsf{Bexpr}\text{“]”}\langle \mathsf{Blist} \rangle \\
\langle \mathsf{Bexpr} \rangle & \rightarrow & \langle \mathsf{Bexpr} \rangle \text{“and”}\langle \mathsf{Bexpr} \rangle \mid \langle \mathsf{Bexpr} \rangle \text{“or”}\langle \mathsf{Bexpr} \rangle \mid \\
 & & \text{“not(”}\langle \mathsf{Bexpr} \rangle \text{“)”} \mid \langle \mathsf{locpath} \rangle \\
\langle \mathsf{axis} \rangle & \rightarrow & \text{“self”} \mid \text{“child”} \mid \text{“parent”} \mid \text{“descendant”} \mid \text{“ancestor”} \mid \cdots
\end{array}
$$

Figure 21: Core XPath

$\mathsf{Bookshop}[\ \textit{not}\ \mathsf{child} :: \mathsf{Author}]$ *and* $\tau(Q_2) = \mathsf{Bookshop} \wedge \neg \mathsf{EX}_{\triangleright}\mathsf{Author}$. $\mathbb{W}$*-queries* Q_3 *and* Q_4*, expressing cyclic conditions, cannot be encoded in Core XPath.*

9.2 UnQL

We show that UnQL database can be immediately mapped to $\mathbb{W}$-instances and how to encode UnQL queries into modal formulae. UnQL is a language for querying data organized as a rooted directed graph with labeled edges [8]. A rooted graph is a pointed graph such that all the nodes are accessible from the root node.[4] We rewrite the UnQL graphs into $\mathbb{W}$-graphs.

Procedure 9.3 *Let* $DB = \langle\langle N, E, \ell\rangle, \nu\rangle$ *be an edge-labeled (UnQL) graph rooted at* ν*. Assume, w.l.o.g., that* $DB \notin \ell(E)$*. We define the* $\mathbb{W}$*-graph* $G' = \langle N', E', \ell'\rangle$ *rooted at* ν *as:*

1. $E' = E$ *is the set of edges;*

[4]According to the definition in [8], a rooted labeled graph is a tuple $G = \langle N, E, s, t, \nu, \ell\rangle$, where $\langle N, E, s, t\rangle$ is a multi-graph, $s, t : E \rightarrow N$ denote the source and the target of each edge, respectively; $\nu \in N$ is the root of the graph, and $\ell : E \rightarrow \mathit{Label} \cup \{\bot\}$ is a labeling function for edges.

	$\tau(\mathsf{self :: nodetest\ Blist})$	$=$	$\tau(\mathsf{nodetest}) \wedge \tau(\mathsf{Blist})$
	$\tau(\mathsf{child :: nodetest\ Blist})$	$=$	$\mathsf{EX}_{\triangleright}(\tau(\mathsf{nodetest}) \wedge \tau(\mathsf{Blist}))$
	$\tau(\mathsf{parent :: nodetest\ Blist})$	$=$	$\mathsf{EX}_{\triangleleft}(\tau(\mathsf{nodetest}) \wedge \tau(\mathsf{Blist}))$
	$\tau(\mathsf{descendant :: nodetest\ Blist})$	$=$	$\mathsf{EF}_{\triangleright}(\tau(\mathsf{nodetest}) \wedge \tau(\mathsf{Blist}))$
locstep	$\tau(\mathsf{ancestor :: nodetest\ Blist})$	$=$	$\mathsf{EF}_{\triangleleft}(\tau(\mathsf{nodetest}) \wedge \tau(\mathsf{Blist}))$
	$\tau(\mathsf{tag})$	$=$	tag
nodetest	$\tau(*)$	$=$	true
	$\tau(\varepsilon)$	$=$	true
Blist	$\tau(\mathsf{[Bexpr]\ Blist})$	$=$	$\tau(\mathsf{Bexpr}) \wedge \tau(\mathsf{Blist})$
	$\tau(\mathsf{Bexpr}\ \frac{\mathsf{and}}{\mathsf{or}}\ \mathsf{Bexpr})$	$=$	$\tau(\mathsf{Bexpr}) \frac{\wedge}{\vee} \tau(\mathsf{Bexpr})$
Bexpr	$\tau(\mathsf{not(Bexpr)})$	$=$	$\neg\, \tau(\mathsf{Bexpr})$
	$\tau(\mathsf{/locpath})$	$=$	$\mathsf{root} \wedge \tau(\mathsf{locpath})$
	$\tau(\mathsf{locpath1 \mid locpath2})$	$=$	$\tau(\mathsf{locpath1}) \vee \tau(\mathsf{locpath2})$
locpath	$\tau(\mathsf{locpath1/locpath2})$	$=$	$\tau(\mathsf{locpath1}) \circ \tau(\mathsf{locpath2})$

Figure 22: Core Xpath translation to CTL

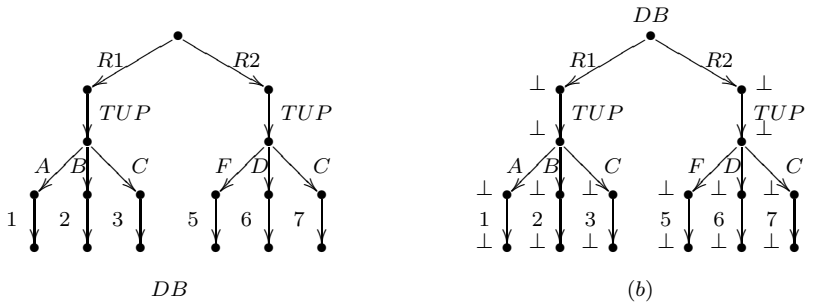


Figure 23: An UnQL graph, called DB, and its translation

2. $N' = N$ *is the set of nodes;*

3. $\ell'(\nu) = DB$*;*

4. $\ell'(m) = \bot$ *if* $m \in N \setminus \{\nu\}$.

G' is rooted since DB is rooted. Thus, we can use the root node (labeled DB) to identify the instance of a database.

For example, the UnQL graph DB in Fig. 23 is rewritten as the $\mathbb{W}$-graph (b) by labeling the set of nodes. The graph (b) can be seen as a Kripke structure $\mathcal{K} = \langle \Sigma, Act, \mathcal{R}, I \rangle$, where Σ is the set of bullet nodes, $Act = \{R1, R1^{-1}, \overline{R1}, \dots\}$, the transition relation R is the set of edges, and the interpretation function $I : \Sigma \to \wp(Act \cup \{\bot, number, \mathsf{true}, \mathsf{false}\})$ is defined as follows:

$$I(s) \;=\; \begin{cases} \{\ell(s), \mathtt{true}\} & \text{if } s \text{ is not a leaf} \\ \{\ell(s), number, \mathtt{true}\} & \text{otherwise} \end{cases}$$

The atom *number* is used to model the fact that in some queries we need to know whether a state of the Kripke structure contains a numeric value (i.e., its label is a number).

Let us consider some examples of UnQL query on the DB database in Fig. 23 and their translations in temporal formulae. We will also use the modal operator B (Before), whose meaning is the following:

Given a state s and a temporal formula ϕ, $s \models \mathsf{AB}_a\phi$ iff for all paths $\pi = \langle \ldots, x, \ell, s, \ldots \rangle$, it holds that $x \models \phi$ and $\ell \in a$, whereas $s \models \mathsf{EB}_a\phi$ iff there is a path $\pi = \langle \ldots, x, \ell, s, \ldots \rangle$ such that $x \models \phi$ and $\ell \in a$.

The expression **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*. The same concept can be expressed by the temporal formula $\varphi_{(1)} = \bot \wedge \mathsf{EB}_{R1}(DB)$ which is true in each dummy (depicted as bullet) state which has as predecessor the root node DB and the edge between them is labeled $R1$, or equivalently, by the *CTL* formula $\psi_{(1)} = \bot \wedge \mathsf{EX}_{R1^{-1}}(DB)$.

The UnQL expression **select** t **where** $\backslash l \Rightarrow \backslash t \leftarrow DB$ retrieves *any edge emanating from the root* and is translated by the formula $\varphi_{(2)} = \bot \wedge \mathsf{EB}_{Act}(DB)$ which is true in all the bullet states accessible by means of a unique edge from the root named DB. The same UnQL query can be translated in *CTL* by means of the formula $\psi_{(2)} = \bot \wedge \mathsf{EX}_{Act}(DB)$.

An UnQL query that look arbitrarily deep into the database to find *all edges with a numeric label* is the following: **select** $\{l\}$ **where** $\_* \Rightarrow \backslash l \Rightarrow \_ \leftarrow DB, isnumber(l)$. This expression can be easily translated into the *CTL* formula $\psi_{(3)} = \bot \wedge number$ which is true in the leaf nodes.

UnQL can look arbitrarily deep into a database; this feature has an immediate translation into *CTL* thanks to the F operator. For example, given the $\mathbb{W}$-instances of Fig. 1, the *CTL* query $\mathsf{Teacher} \wedge \mathsf{EF}_{\mathcal{A}ct_G}\mathsf{Databases}$ requires to find Teacher nodes having a path that eventually reaches a Databases node. It would be interesting to include this feature also into the $\mathbb{W}$-query language; one possibility is to use dotted edges to represent paths of an arbitrary length. The UnQL query

$$\begin{array}{ll} \textbf{select} & \{Tup \Rightarrow \{A \Rightarrow x, D \Rightarrow z\}\} \\ \textbf{where} & R1 \Rightarrow Tup \Rightarrow \{A \Rightarrow \backslash x, C \Rightarrow \backslash y\} \leftarrow DB, \\ & R2 \Rightarrow Tup \Rightarrow \{C \Rightarrow \backslash y, D \Rightarrow \backslash z\} \leftarrow DB \end{array}$$

computes *the join of $R1$ and $R2$ on their common attribute C and the projection onto A and D*. It is easy to check that its application to the database DB has an empty result.

UnQL queries expressing a join condition on a graph do not have a corresponding temporal formula, because in propositional temporal logics it is possible to identify a state only by the set of atomic predicates that are true in it. The nodes with the same properties cannot be distinguished. An intuitive translation for the former query could be the following:

$$\begin{array}{rl} \varphi_{(4)} = & \bot \wedge \mathsf{EB}_{Act}(\bot \wedge \mathsf{EB}_A(\bot \wedge \mathsf{EX}_C(\bot \wedge \mathsf{EX}_{Act}(\bot)))) \vee \\ & \bot \wedge \mathsf{EB}_{Act}(\bot \wedge \mathsf{EB}_D(\bot \wedge \mathsf{EX}_C(\bot \wedge \mathsf{EX}_{Act}(\bot)))) \end{array}$$

The formula φ holds in two states of the $\mathbb{W}$-graph (b) in Fig. 23 (the bullet nodes with unique ingoing edge labeled 1 and 6 respectively), and thus, the result of its application on the database DB is not correct.

Example 9.4 *Let us consider again the example of Fig.* 16*. The graph-based representation of the XML document (see Fig.* 14*) can be translated into an UnQL graph named, for example, XMLdoc, which has labeled edges (and nodes without labels). $\mathbb{W}$-queries of Fig.* 16 *can be translated into UnQL as follows.*

$$\begin{array}{rll} Q_1 = & \textbf{select} & \mathsf{Bookshop} \\ & \textbf{where} & \mathsf{Bookshop} \Rightarrow [\hat{}\mathsf{Author}]^* \leftarrow\!\leftarrow \mathsf{XMLdoc}, \\ & & \mathsf{Author} \Rightarrow \{\} \leftarrow\!\leftarrow \mathsf{XMLdoc} \end{array}$$

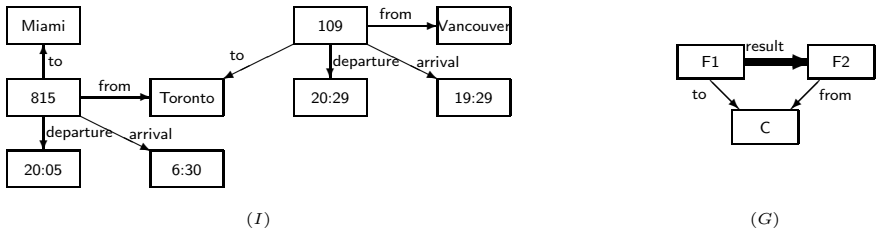


Figure 24: GraphLog representation of a database and a query

Query (Q_1) *retrieves (at arbitrary depth) any path in the instance graph starting with a* Bookshop *edge and not containing the label* Author. *Moreover,* Author *edge must exists anywhere in the database.*

$Q_2 =$ **select** Bookshop
where Bookshop $\Rightarrow [\hat{}\mathsf{Book}]^* \leftarrow\!\leftarrow$ XMLdoc

Query (Q_2) *retrieves (at arbitrary depth) any path in the instance graph any path starting with a* Bookshop *edge and not containing the label* Book.

$Q_3 =$ **select** x
where $x = \{\mathsf{Author} \Rightarrow \{\mathsf{Name}, x\}\} \leftarrow\!\leftarrow$ XMLdoc

Query (Q_3) *retrieves (at arbitrary depth) cycles in the instance graph composed by a (cyclic) edge labeled* Author *having an outgoing* Name *edge.*

$Q_4 =$ **select** x
where $x = \{\mathsf{Author} \Rightarrow x\} \leftarrow\!\leftarrow$ XMLdoc

Query (Q_4) *retrieves (at arbitrary depth) cycles in the instance graph composed by a (cyclic) edge labeled* Author.

To sum up, join-free queries of UnQL can be solved using CTL model-checking.

9.3 GraphLog

GraphLog is a query language based on a graph representation of both data and queries [15]. Queries represent graph patterns corresponding to paths in databases.

For example, the graph I in Fig. 24 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. [5] GraphLog databases are not required to be rooted and therefore the corresponding $\mathbb{W}$-graphs are not necessarily rooted.

In GraphLog 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 [15] is a directed labeled multigraph with distinguished edges

[5] According to the definition in [15] a *directed labeled multigraph* G is a tuple $\langle N, E, L_N, L_E, \iota, \nu, \epsilon\rangle$ where N is a finite set of nodes, E is a finite set of edges, L_N is a set of node labels, L_E is a set of edge labels, ι, the incidence function, is a function from E to N^2 that associates with each edge in E a pair of nodes from N, ν, the node labeling function, is a function from N to L_N that associates with each node in N a label from L_N, and finally ϵ, the edge labeling function, is a function from E to L_E that associates with each edge in E a label from L_E.

(they define new relations that will hold, after the query application, between two objects in a chosen database whenever they satisfy the requirements of the query graph), without isolated nodes, and having the following properties:

1. the nodes are labeled by sequences of variables;
2. 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;
3. the distinguished edge can only be labeled by a positive non-closure literal.

For example, graph (G) in Fig. 24 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-constructive part of this query is:

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

(which intuitively 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. 25 representing the relationships between people. Their natural translation into *CTL* should be:

$$\begin{aligned}
\Psi(Q_1) \quad &= \quad \mathsf{Person} \wedge ((\mathsf{EX}_{\mathsf{parent}}\mathsf{Person}) \rightarrow \\
&\qquad (\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{parent}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}^{-1}}\mathsf{Person}))) \\
\Psi(Q_2) \quad &= \quad \mathsf{Person} \wedge ((\mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}\mathsf{Person})) \rightarrow \\
&\qquad (\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}}(\mathsf{Person} \wedge \mathsf{EX}_{\mathsf{relative}^{-1}}\mathsf{Person}))))
\end{aligned}$$

It is easy to see that the node n of the instance graph $\mathcal{I}$ in Fig. 25 satisfies the formula $\Psi(Q_1)$ but does not satisfy the natural interpretation of relationships between people. The problem is that in *CTL* it is not possible to constrain the last unary predicate Person in $\Psi(Q_1)$ to be satisfied by the same state of the first unary predicate Person.

In order to overcome this limitation and to be able to apply model-checking techniques to GraphLog, one should first 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 all labeled a in the original graph (see [17] 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.

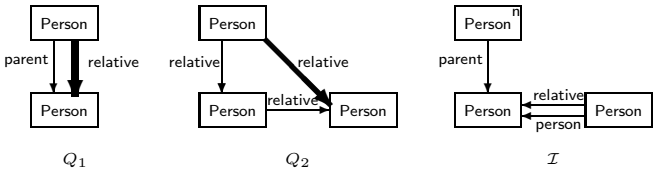


Figure 25: Two query graphs and an instance

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

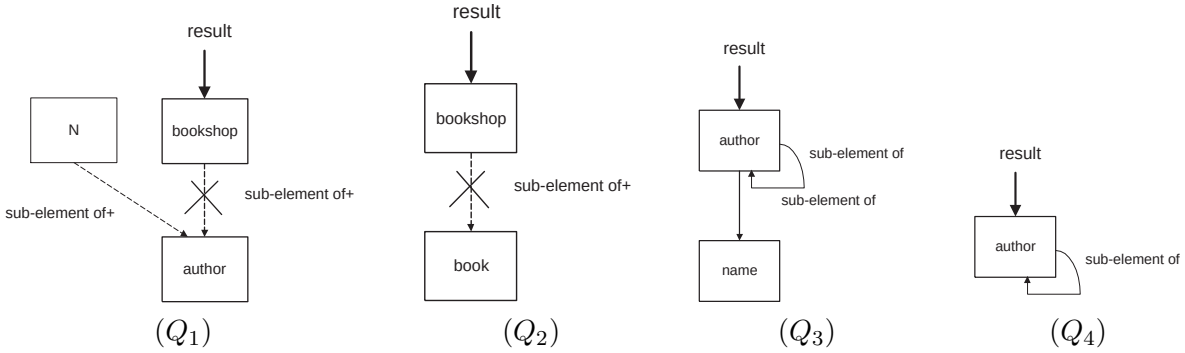


Figure 26: Graphlog queries to (Q_1) *find Bookshop nodes which do not have an Author descendant (but an Author node must exist in the database)* (Q_2) *find Bookshop nodes which do not have a Book descendant* (Q_3) *find Author nodes with a Name child and which do not have a self reference* (Q_4) *find Author nodes which do not have a self reference.*

Example 9.5 $\mathbb{W}$*-queries of Fig.* 16 *can be correctly translated into GraphLog as shown in Fig.* 26*. The* result *literal is the literal labeling the distinguished edges. In the Graphlog representation of query* Q_1 *we want to find Bookshop nodes which do not have an Author descendant (the negative closure literal* $\neg$*sub-element of* $+$ *is represented by crossing over the edge labeled with it), but an Author node must exist in the database has a descendant of a generic node, which is labeled with the variable* N*. In queries* Q_3 *and* Q_4 *we want to find* author *nodes which have a self reference.*

9.4 G-log

G-Log [36] as GraphLog, is a query language in which data, queries, and generic rules are expressed by graphs. As we have already pointed out in this work, $\mathbb{W}$ is inspired to non-constructive semantics of G-log. 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 than $\mathbb{W}$-ones and are characterized by three colors: *red solid* specifies positive conditions, *red dashed* specifies negative conditions, while *green* indicates a desired situation in the resulting instance. Nevertheless, often G-Log queries has only one green node with an edge pointing to a red solid node ν. These queries are the same as $\mathbb{W}$-queries, in which ν is the node pointed by the query where the color attribute red solid (dashed) is simply solid (dashed). To be more precise, we do not consider that G-Log query application modifies original instances by (possibly) adding nodes pointing to sub-graphs satisfying constraints specified in the red part of a given query; in fact, we manage the problem of retrieving nodes satisfying the constraints visually rendered in the red part of the query. Moreover, in G-Log two kinds of nodes are allowed, complex and atomic nodes, but this feature can be simulated by modifying the co-domain of the *label* function in $\mathbb{W}$. Instead, solid edges are forbidden to enter into dashed nodes in G-Log. This feature of $\mathbb{W}$ is one of the main points for programming nesting quantifications 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 satisfies this further requirement, is a G-Log query.

Semantics of query application is given in [36] via the notion of *graph embedding* and

in [16] using the notion of *subgraph bisimulation* (both NP complete, also without considering negation in query graphs). It is immediate to verify that G-Log queries that belong to the four families of graphs defined in Section 6.3 can be correctly solved using the model-checking approach. This result is basically the same proved in Theorem 6.11, extended to the simple cyclic queries allowed. We have already discussed (Sect. 7) that for complexity limits we cannot extend the method to all cyclic queries.

10 Datalog interpretation and stratification

In this section we relate the CTL interpretation of $\mathbb{W}$-queries with their logic programming interpretation. We use a fragment of Prolog without arithmetic functions, without functional symbols, but allowing constant symbols and negation in the clause bodies. This is a subset either of Prolog or of Datalog with negation. In the remaining part of the section we will refer this sublanguage simply as Datalog.

A $\mathbb{W}$-instance can be naturally represented as a set of Datalog facts. A $\mathbb{W}$-query, instead, by a Datalog program with negation. The latter encoding can be obtained from the CTL-semantics by adapting Lloyd-Topor transformation ([44], dealing with first-order logic encoding) to (propositional) CTL formulas. Acyclic queries originate stratified programs, while for cyclic queries this is not ensured. However, stable models (see e.g., [43]) of those programs can be computed by using answer set solvers ([45]), but it is known that establishing the existence of stable models is NP-complete, while our method for solving queries is polynomial.

10.1 Instance Encoding

A $\mathbb{W}$-instance I can be naturally rendered by a set of Datalog facts.

- For each node n of the instance, add a fact: `node(id_node_n,label)` where `id_node_n` is the identifier of the node n and `label` is its label (namely $\ell_{\mathcal{L}}(n)$).
- For each edge $\langle m, \langle solid, label\rangle, n\rangle$ of I, add a fact: `edge(id_node_m,in_node_n,label)`, where `id_node_m` and `id_node_n` are the identifiers of the nodes m and n, respectively.

As an example, the $\mathbb{W}$-instance I of Fig. 1 can be rendered by the Datalog program $P_{\mathcal{I}}$ (strings do not start with capital letters according to logic programming syntax restrictions):

```
node(1,teacher).    node(5,37).          edge(1,3,teaches).    edge(3,7,cname).
node(2,teacher).    node(6,40).          edge(1,5,age).        edge(4,3,attends).
node(3,course).     node(7,databases).   edge(2,3,teaches).    edge(4,8,name).
node(4,student).    node(8,smith).       edge(2,6,age).
```

11 Query encoding

The main idea behind the query encoding we are going to propose is to use `node` and `edge` relations and first-order quantifiers to encode the notion of path.

With abuse of notation, let us denote with m, n etc. either nodes of the KTS representing the instance $\mathcal{I}$ or their identifiers in $P_{\mathcal{I}}$. We would like that:

- The formula $p \wedge \mathsf{EX}_a \varphi$ holds in m if and only if the formula $\mathtt{node}(m, p) \wedge \exists n\,(\mathtt{edge}(m, n, a) \wedge \tau(\varphi))$, where $\tau(\varphi)$ is the translation of the formula φ, is a logical consequence of P_I.
- The formula $p \wedge \mathsf{AX}_a \varphi$ holds in m if and only if the formula $\mathtt{node}(m, p) \wedge \forall n\,(\mathtt{edge}(m, n, a) \rightarrow \tau(\varphi))$ is a logical consequence of P_I.

As an example, the four queries (R_1)–(R_4) of Figure 4 can be translated as follows:

- Rule $(R1)$: $\mathsf{Teacher} \wedge \mathsf{EX}_{\mathsf{teaches}}(\mathsf{Course})$ can be mimicked by

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \exists \mathtt{ID2}\,(\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \wedge \mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

- Rule $(R2)$: $\mathsf{Teacher} \wedge \mathsf{EX}_{\overline{\mathsf{teaches}}}(\mathsf{Course})$ has the same meaning as

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \exists \mathtt{ID2}\,(\neg\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \wedge \mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

- Rule $(R3)$: $\mathsf{Teacher} \wedge \mathsf{AX}_{\mathsf{teaches}}(\neg\mathsf{Course})$ can be mimicked by:

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \forall \mathtt{ID2}\,(\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \rightarrow \neg\mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

  which is equivalent to:

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \forall \mathtt{ID2}\,(\neg\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \vee \neg\mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

- Finally, rule $(R4)$: $\mathsf{Teacher} \wedge \mathsf{AX}_{\overline{\mathsf{teaches}}}(\neg\mathsf{Course})$ is equivalent to

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \forall \mathtt{ID2}\,(\neg\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \rightarrow \neg\mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

  which is equivalent to:

$$\mathtt{node}(\mathtt{ID1}, \mathtt{teacher}) \wedge \forall \mathtt{ID2}\,(\mathtt{edge}(\mathtt{ID1}, \mathtt{ID2}, \mathtt{teaches}) \vee \neg\mathtt{node}(\mathtt{ID2}, \mathtt{course}))$$

First-order quantifiers are implicitly definable in a Datalog program. Variables occurring in the body of a clause can be seen as existentially quantified. Negation of atoms based on those symbols therefore define universally quantified variables. Disjunction is introduced when more clauses define the same predicate. Let us consider the following simple example:

```
p(X) :- q(X,Y), not r(X,Y), not s(X).
s(X) :- t(X,Z).
s(X) :- v(X)
```

`s(X)` holds if exists `Z` s.t. `t(X,Z)` holds or if `v(X)` holds. `p(X)` holds if exists `Y` s.t. `q(X,Y)` holds, and `r(X,Y)` and `s(X)` do not hold. Combining the two clauses, we have

$$\mathtt{p}(\mathtt{X}) \leftarrow \exists \mathtt{Y}\,(\mathtt{q}(\mathtt{X}, \mathtt{Y}) \wedge \neg\mathtt{r}(\mathtt{X}, \mathtt{Y})) \wedge (\mathtt{v}(\mathtt{X}) \vee \forall \mathtt{Z}\,\neg\mathtt{t}(\mathtt{X}, \mathtt{Z}))$$

(since $\neg\exists \mathtt{Z}\mathtt{t}(\mathtt{X}, \mathtt{Y})$ is equivalent to $\forall \mathtt{Z}\,\mathtt{t}(\mathtt{X}, \mathtt{Y})$).

Stratification, namely an ordering of the predicate symbols of the program that ensures that no predicate is defined in terms of its negation, is an important property for Datalog programs. The above program is stratified, while `u(X):- not u(X).` is not. Stratified programs have a deterministic semantics. Unstratified programs may allow several models. Stable models are proposed as "good" models for those programs [43]. However, establishing the existence of a stable model is an NP-complete problem.

In Definition 6.5 we associate a CTL formula to a $\mathbb{W}$-query. Here, we use the ideas of Lloyd-Topor transformation to associate a Datalog program $P(Q)$ to a given query $Q = \langle G, \nu \rangle$. For each node n of the query, we introduce the unary predicate symbol `query_n` and, possibly, other unary predicate symbol `aux_n`, `aux_n_1`, `aux_n_2`, ...

- Each solid node n, with label `label` without outgoing edges is rendered by the rule `query_n(ID) :- node(ID,label).`
  If the node is dummy, by `query_n(ID) :- node(ID,_).`

- Each dashed node n, with label `label` without outgoing edges is rendered by the rules `query_n(ID) :- not node (ID,label).`
  If the node is dummy, by `query_n(ID) :- not node(ID,_).`

- Let us consider a solid node ν (`nu`) with outgoing edges as in Figure 5. We introduce the rules

```
query_nu(ID) :- node(ID,label_of_nu),
   (op) edge(ID,ID1,e_1), query_b1(ID1),
      ...
   (op) edge(ID,IDh,e_h), query_bh(IDh),
   not aux_nu_1(ID'1),...,not aux_nu_k(ID'k).
aux_nu_i(ID) :- %%% i = 1,...,h
   (nop) edge(ID,ID1,e'_1), query_ci(ID1).
```

  where `(op)` is omitted if the edge is solid and is `not` if the edge is dashed. `(nop)` is the opposite of `op`.

- Let us consider a dashed node ν (`nu`) with outgoing edges as in Figure 5. We introduce the rules

```
query_nu(ID) :- not node(ID,label_of_nu).
query_nu(ID) :- %%% i = 1,...,h
   not aux_nu_i(ID).
query_nu(ID) :- %%% j = 1,...,k
   (op) edge(ID,ID1,e'_j),
   query_cj(ID1).
aux_nu_i(ID) :- %%% i = 1,...,h
   (nop) edge(ID,ID1,e_i), query_bi(ID1).
```

  where `(op)` is omitted if the edge is solid and is `not` if the edge is dashed. `(nop)` is the opposite of `op`.

As an example, consider again the queries $(R1)$–$(R4)$ of Figure 4.

```
%%% P(R1)
query_1(ID) :- node(ID,teacher),edge(ID,teaches,ID2),query_2(ID2).
query_2(ID) :- node(ID,course).
%%% P(R2)
query_1(ID) :- node(ID,teacher),not
edge(ID,teaches,ID2),query_2(ID2). query_2(ID) :- node(ID,course).
%%% P(R3)
query_1(ID) :- node(ID,teacher), not aux_1(ID). aux_1(ID) :-
edge(ID,teaches,ID2), query_2(ID2). query_2(ID) :- node(ID,course).
%%% P(R4)
query_1(ID) :- node(ID,teacher), not aux_1(ID). aux_1(ID) :-  not
edge(ID,teaches,ID2), query_2(ID2). query_2(ID2) :- node(ID,course).
```

Let us observe, for instance, that $P_{\mathcal{I}} \cup P(R1) \models \texttt{query\_1}(1) \wedge \texttt{query\_1}(2)$.

Proposition 11.1 *Acyclic $\mathbb{W}$-queries generate stratified Datalog programs.*

Proof. Immediate from the fact that no recursion is introduced. □

Proposition 11.2 *For the queries when $\Psi_\nu(G)$ is defined, a state n is such that $n \models \Psi_\nu(G)$ if and only if $\mathtt{query}_\nu(n)$ is a logical consequence of $P_\mathcal{I} \cup P(Q)$.*

Proof. Immediate from the first-order encoding of the notion of path in a KTS. □

12 Related Work

A work which introduces temporal-logic queries for model understanding and model-checking is [12]. The author proposes an approach for inferring properties of models as a special case of evaluating temporal-logic queries by using a model-checker. In [12] a query is a *CTL* formula with a special symbol ?, called placeholder, which appears exactly once. Given a model M, the semantics of a query is a proposition p such that replacing ? with p in the query, it results a formula that holds in M and is as *strong* as possible. Consider for example the $\mathbb{W}$-instance of Fig. 1. The evaluation of the query $\mathsf{AG}_{\mathcal{A}ct}$? on that $\mathbb{W}$-graph, gives as result $p = \text{Teacher} \vee \text{Course} \vee \text{Databases} \vee \text{Student} \vee \text{Smith} \vee 40 \vee 37$, i.e., an indication of all the labels included in the model of Fig. 1. Whereas the evaluation of the query $\text{Teacher} \wedge \mathsf{EX}_{\mathcal{A}ct}$? on the same model gives as result all the successors of Teacher states, i.e., $p = \text{Course} \vee 40 \vee 37$.

Another work which addresses the task of computing in an efficient way the information retrieval problem for structured document is [34]. The authors propose an approach for expressing unary queries, i.e., queries that map a tree into a set of its nodes, by means of tree automata (called query automata) and apply them to either ranked or not ranked trees corresponding to structured databases. In [34] the expressive power of query automata is proved to be equivalent to monadic second order logic.

In recent works by Gottlob et al., fragments of XPath queries (Full XPath1 queries) are shown to be computable in polynomial time w.r.t. the sizes of the query and of the documents [24, 26, 27]. Recent works [30, 5] study the encoding of XPath queries into CTL, enlarging the classes previously encoded.

Other papers who focus on querying as a matching problem are [31, 40]. In [31] the authors consider a class of XPath queries that contain branching, label wildcards and can express descendant relationships between nodes and study the complexity of the containment problem. In particular, they prove that this fragment of XPath has an intractable containment problem in general, but they provide an algorithm using tree automata that runs in PTIME in most of the cases of practical interest. They also state that the fragment of the logic *CTL*, which consists of *true*, conjunction, $\mathsf{EX}\phi$ and $\mathsf{EF}\phi$ formulas, is sufficient for the tree patterns they consider. As far as acyclic $\mathbb{W}$-solid queries are concerned, our approach can be considered an instance of the containment problem studied in [31]. According to their notation, we consider the case where the first pattern is in the set of patterns representing semistructured instances which contain only branching and descendant relationships, and the second pattern is a query which can be expressed by using XPath as well. However, in this paper we allow queries to include negative requirements (using dashed nodes and edges) and thus we need a larger fragment of *CTL* including the $\mathsf{EG}\phi$ formula and the universal modality A.

In this context, when query answering is a matching problems between trees, it is essential to identify redundant parts of queries, in order to reduce their size. In [40] the authors study tree pattern minimization problem both in the absence and in the presence of integrity constraints on the underlying tree-structured database. This proposal can be applied also to XML documents whenever queries are expressed with tree shaped structures, e.g. with the XPath fragment studied in [31].

13 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 methods coming from the model-checking community. The *graph-formula* translation could be extended to a wider family of graphs including join conditions and generic cycles by using Hybrid Logics: the modal logics semantics forces a bisimulation equivalence between subgraphs satisfying 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 and cyclic queries. However, the bisimulation based semantics allows one to simplify queries and data by using minimization modulo bisimulation that can be done in linear time on acyclic graphs and for some families of cyclic graphs [19].

We have tested 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 [18] in a slight different context.

Our work is included in the project analyzing graph-based query languages (G-log [36], WG-log [16]) that recently addressed the problem of querying XML-specified information (XQuery [7]). Our results can be immediately used by all these languages.

References

[1] S. Abiteboul. Querying semi-structured data. In F. N. Afrati and P. G. Kolaitis eds., In *Proc. of the International Conference on Database Theory (ICDT'97)*, Vol. 1186 of *LNCS*, pp. 1–18, 1997.

[2] S. Abiteboul, D. Suciu, and P. Buneman. Data on the Web: from Relations to Semistructured Data and XML. *Morgan Kaufmann Series in Data Management Systems*, 2002.

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

[4] P. Aczel. *Non-well-founded sets.* Vol. 14 of *Lecture Notes, Center for the Study of Language and Information.* Stanford, 1988.

[5] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL Model Checking for Processing Simple XPath Queries. In *Proc. of 11th International Symposium on Temporal Representation and Reasoning (TIME 2004)*, IEEE, pp. 117–124, 2004.

[6] P. Blackburn, and J. Seligman. What are hybrid languages? In *Advances in Modal Logic*, Volume 1, pp. 41–62, CSLI Publications, Stanford, 1996.

[7] D. Braga and A. Campi. A Graphical Environment to Query XML Data. In *4th International Conference on Web Information System Engineering (WISE 2003)*, IEEE Computer Society, pp. 31–40, 2003.

[8] 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 International Conference on Management of Data*, pp. 505–516, 1996.

[9] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. Adding structure to unstructured data. In *Proc. of the International Conference on Database Theory (ICDT'97)*, Vol. 1186 of *LNCS*, pp. 336–350, 1997.

[10] L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. In *Proc. of the 10th European Symposium on Programming (ESOP 2001)*, Vol. 2028 of *LNCS*, pp. 20–28, 2001.

[11] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a graphical language for querying and restructuring XML documents. In Vol. 31 of *Computer Networks*, pp. 1171–1187, 1999.

[12] W. Chan. Temporal-logic Queries. In *Proc. of 12th International Conference on Computer Aided Verification (CAV 2000)*. Vol. 1855 of *LNCS*, pp. 450–463. Chicago, USA, 2000.

[13] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. *Proc. of 11th International Conference on Computer Aided Verification (CAV 1999)*. Vol. 1633 of *LNCS*, pp. 495–499, 1999.

[14] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specification. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[15] M. P. Consens and A. O. Mendelzon. Graphlog: a Visual Formalism for Real Life Recursion. In *Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Priciples of Database Systems*, pp. 404–416, 1990.

[16] A. Cortesi, A. Dovier, E. Quintarelli, and L. Tanca. Operational and Abstract Semantics of the Query Language G-Log. *Theoretical Computer Science*, 275(1–2):521–560, 2002.

[17] E. Damiani and L. Tanca. Blind Queries to XML Data. In *Proc. of the 11th International Conference Database and Expert Systems Applications*, Vol. 1873 of *LNCS*, pp. 345–356, 2000.

[18] L. de Alfaro. Model Checking the World Wide Web. In *Proc. of 13th Conference on Computer Aided Verification (CAV 2001)*, Vol. 2102 of *LNCS*, pp. 337–349, 2001.

[19] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. In Theoretical Computer Science 311(1–3):221–256, 2004.

[20] A. Dovier and C. Piazza. The Subgraph Bisimulation Problem. IEEE Transactions on Knowledge and Data Engineering 15(4):1055–1056, 2003.

[21] A. Dovier and E. Quintarelli. Model-checking based data retrieval. In G. Ghelli and G. Grahne, eds., In *Proc. of the 8th Biennial Workshop on Data Bases and Programming Languages (DBPL'01), Revised Papers*. Vol. 2397 of *LNCS*, pp. 62–77, 2002.

[22] 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.

[23] 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.

[24] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *Proc. of 28th International Conference on Very Large Data Bases (VLDB 2002)*, pp. 95–106, 2002.

[25] G. Gottlob and C. Koch. Monadic Queries over Tree-Structured Data. In *Proc. of 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pp. 189–202, 2002.

[26] G. Gottlob, C. Koch, and R. Pichler. XPath Query Evaluation: Improving Time and Space Efficiency. In *Proc. of the 19th International Conference on Data Engineering (ICDE 2003)* pp. 379–390, 2003.

[27] G. Gottlob, C. Koch, and R. Pichler. XPath Processing in a Nutshell. SIGMOD Record 32(2):21–27, 2003.

[28] G. Gottlob, C. Koch, and R. Pichler. The Complexity of XPath Query Evaluation. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2003.

[29] 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.

[30] M. Marx. First order paths in ordered trees. In *Proc. of ICDT 2005, Tenth International Conference on Database Theory*, Edinburgh, 2005.

[31] G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 65–76, 2002.

[32] R. Milner. A Calculus of Communicating Systems. Vol. 92 of *LNCS*, Springer-Verlag, Berlin, 1980.

[33] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking. A tutorial introduction. In *Proc. of the International Static Analysis Symposium SAS'99*. Vol. 1694 of *LNCS*, pp. 330–354, 1999.

[34] F. Neven and T. Schwentick. Query Automata. In *Proc. of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 205–214, 1999.

[35] R. Paige and R. E. Tarjan. Three Partition refinements algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[36] J. Paredaens, P. Peelman, and L. Tanca. G–Log: A Declarative Graphical Query Language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):436–453, 1995.

[37] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, Number 104 of *LNCS*. Springer Verlag, 1980.

[38] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying Semistructured Heterogeneus Information. In *Proc. of the International Conference on Deductive and Object-Oriented Databases (DOOD'95)*, pp. 319–344, 1995.

[39] M. Y. Vardi. The Complexity of relational Query Languages. In *Proc. of ACM Symposium on Theory of Computing (STOC'82)*, pp. 137–146, 1982.

[40] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *Proceedings of the Twentieth ACM SIGMOD International Conference on Management of Data*, 2001.

[41] World Wibe Web Consortium. XML Path Language (XPath) Reccomendation. `http://www.w3.org/TR/xpath`, Nov. 1999.

[42] The ARANEUS Project. `http://www.dia.uniroma3.it/Araneus`.

[43] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *ICLP*, pages 1070–1080, MIT Press, 1988.

[44] J. W. Lloyd and R. W. Topor. Making Prolog more Expressive. Journal of Logic Programming 3:225–240, 1984.

[45] Web references for some ASP solvers. ASSAT: `assat.cs.ust.hk`. CCalc: `www.cs.utexas.edu/users/tag/cc`. CModels: `www.cs.utexas.edu/users/tag/cmodels`. DeReS and aspps: `www.cs.uky.edu/ai`. DLV: `www.dbai.tuwien.ac.at/proj/dlv`. SModels: `www.tcs.hut.fi/Software/smodels`.