Querying Web Forms and Nested Semi-structured Data

Mourad Ykhlef[*]

Abstract. *Semi-structured data are commonly represented by labelled graphs. These graphs can be simple or nested. In this paper we present how to model nested semi-structured data in the presence of Web forms. Our motivation is to bring our data model more realistic to capture the richness of Web data. The main purpose of the paper is to provide a mechanism to query nested semi-structured data and web forms in a uniform way.*

***Keywords**: Nested Semi-structured Data, Web Forms, Query embedding, Calculus and Logic*

1. Introduction

Semi-structured data [3, 11, 4, 5, 6, 14] are commonly represented by labelled db-graphs where nodes are atomic (i.e., string, integer, ...). The problem of a db-graph, it can not model semi-structured data when data are nested. To overcome this problem and to make web modelling more realistic, we have proposed nested db-graph model [6]. In nested db-graph the nodes can be labelled by db-graphs (see figure 1). The importance of nested model is similar to the importance of nested tables in relational model [2]. In this paper we show how nested semi-structured data, in the presence of web form, can be modelled and queried. A Web form, like database view, is used to limit the type of queries that can be posed on the Web. The problem addressed here is how to traduce a query posed on semi-structured data and web forms to a set of queries posed on the nested db-graph that represents the semi-structured data as well as the data abstracted by web forms. Some related works exist in the literature. A representation of Web data as complex objects can be found in [12]. The authors of [14] propose a model based on nested tables for modelling semi-structured data. The language proposed in [14] can not query data at any level of nesting except if they know the depth of nesting. Up to our knowledge, few works have addressed the web forms. One important work is given in [10, 11]. The problem of their proposal is the absence of the nesting in their model. The user can not query the web if (s)he did not know what a form should return exactly.

The rest of this paper is organized as follows. Section 2 is devoted to the presentation of our (nested) semi-structured data model. Section 3 introduces a parameterized graph query language Graph($\mathcal{L}$) where $\mathcal{L}$, the parameter, is a path query language. We instantiate $\mathcal{L}$ by a path query language based on nested path formulas called Nest-Path. The main problem arising while using Graph(Nest-Path) to query a nested db-graph is that it requires to have knowledge of the levels of nesting. The introduction of a new instantiation of $\mathcal{L}$ called Emb-Path solve this problem. Section 4 show how the web forms can be modelled in our framwork. In section 5 we investigate how user can query semi-structured

[*]Collge of Computer and Information Sciences, department of Information Systems, King Saud University, Riyadh, Saudi Arabia, email: ykhlef@ccis.ksu.edu.sa

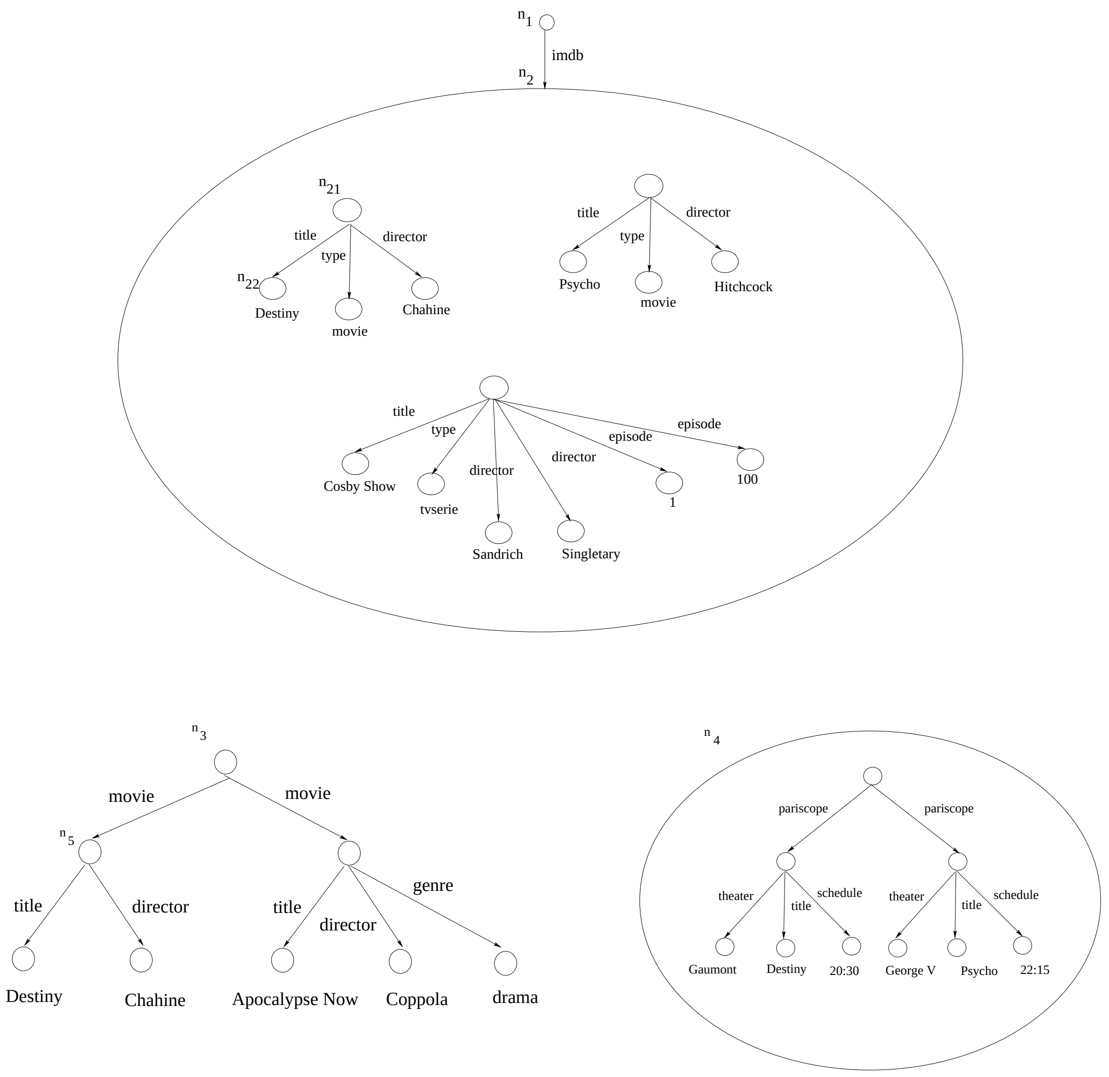

Figure 1. A nested db-graph.

data in the presence of web forms. Further research topics conclude the paper.

2. Data model

Nested db-graphs (db-graphs for short) are a generalization of flat db-graphs in the sense that some nodes are allowed "to be" themselves db-graphs. Hence, in the next definition, we shall distinguish two sets of nodes: a set N_s of simple nodes and a set N_c of complex nodes.

Definition 1 (Nested db-graph) *Let $\mathcal{V}$ be an infinite set of atomic values. Let N_s be a set of simple nodes and N_c a set of complex nodes. Let E be a set of edges (identifiers). Let org and $dest$ be functions assigning to an edge respectively its origin and its destination. Let λ be a polymorphic labelling function for nodes and edges such that labels of simple nodes and edges are atomic values in $\mathcal{V}$ whereas labels of complex nodes are db-graphs. A nested db-graph $G = (N_s, N_c, E, \lambda, org, dest)$ is defined recursively by:*

1. *If N_c is empty then $G = (N_s, N_c, E, \lambda, org, dest)$ is a nested db-graph. In this case, G is called a flat db-graph.*

2. *$G = (N_s, N_c, E, \lambda, org, dest)$ is a nested db-graph over the db-graphs $G_1, .., G_k$ if for each $n \in N_c$, $\lambda(n)$ is one of the db-graphs $G_1, .., G_k$*

Example 1 *Figure 1 is an example of a nested db-graph. It contains two complex nodes n_2 and n_4. The destination n_2 of the edge labelled by* `imdb`*, contains a db-graph providing information about movies and tv-series. The complex node n_4 is labelled by a db-graph representing a relational database having a single relation $pariscope(theater, title, schedule)$. The simple node n_3 is the* root *of a flat db-tree giving information about movies too.*

3. Querying semi-structured data without forms

3.1. Querying non-nested semi-structured data

In the literature, Most of the languages proposed for querying semi-structured data represented by a graph are based on a similar approach, although the paradigms used are different. A query is meant to extract sub-graphs from the initial graph or equivalently the roots of these sub-graphs. What are the criteria participating in the specification of the sub-graph retrieval? Essentially, the languages allow one to express two kinds of properties: *(i)* reachability of the root of the output sub-graphs via a specific path; *(ii)* existence, in the output sub-graph, of some specific paths. It is easy to see that both kinds of property rely on some paths. This explains why most languages like ours [4, 5] are based on an intermediate language which retrieves paths in a graph. These path languages often use path expressions as their basic constructs. Their expressive power determines the expressive power of the semi-structured query language based on it. In the following, for the purpose of the presentation, We will review our semi-structured $\mathcal{L}_{path}$ and Graph($\mathcal{L}_{path}$) [4, 5]. The languages considered in this paper are calculus. Thus we suppose that four sorts of variables are on hand: path and graph variables denoted by $X, Y, Z, \ldots$, label variables denoted by $x, y, z, \ldots$ and data variables denoted by $\alpha, \beta, \gamma, \ldots$ We sometimes use bold capital characters $\mathbb{X}, \mathbb{Y}, \ldots$ when the sort of the variable is irrelevant. A *term* is defined as usual.

A path formula φ of $\mathcal{L}_{path}$ may have free variables. All free variables of φ are forced to be of the sort path. Defining the semantics of $\mathcal{L}_{path}$ is done by defining the relation $G,\nu \models \varphi$ for any db-graph G and any valuation ν of the free variables in φ.

Definition 2 *A query in the language Graph($\mathcal{L}_{path}$) is an expression of the form $\{X_1, \ldots, X_n \mid \phi\}$ where X_i's are graph variables and moreover X_i's are the only free variables of the graph formula ϕ. An atomic graph formula is an expression of one of the following forms:*

1. *A path formula in $\mathcal{L}_{path}$,*

2. *$t : X$ where t is a graph term and X is a path variable,*

3. *$X[t]$ where X is a path variable and t is a graph term,*

4. *$t_1 \sim t_2$ where both t_1 and t_2 are graph terms and where $\sim$ is the symbol of bisimulation [13].*

A general graph formula is defined in the usual way by introducing connectives and quantifiers.

Roughly speaking, $t : X$ means that X is a path whose origin is a root of the graph t. Hence the formula of the kind $t : X$ expresses what has been formerly introduced as a retrieval condition of the kind *(i) existence, in the output sub-graph, of some specific path*.

Intuitively, the atom $X[t]$ checks whether the graph t is rooted at the destination of the path X. This formula illustrates what we have previously introduced as a retrieval condition of the kind *(ii) reachability of the root of the output sub-graphs via a specific path*.

The formal definitions of graph formula satisfaction and of graph query answering are not developed here. The interested reader can find these definitions in [5]. In the next section, we review our two nested db-graph query languages. The first one is based on a path language called Nest-Path and the second one is based on Emb-Path, for more details see [6].

3.2. Nest-Path: a nested path calculus

The language Nest-Path is a path expression language which extends Path by nesting (simple) path expressions. A path expression in Path is an abstraction for describing paths which have a level of nesting equal to 0. Here we need a mechanism to abstract paths belonging to any level of nesting. Let us first illustrate the language Nest-Path.

A (simple) path expression (in Path) can be of the form $x \triangleright \alpha$ and then the intention is to describe all paths reduced to a single edge labelled by x whose destination is a *simple* node labelled by α. We extend these simple expressions by considering the more general form $l \triangleright q$ where q is a path query. Such a nested path expression is meant to specify paths reduced to a simple edge labelled by l and whose destination is a *complex* node on which the query q should evaluate to a non-empty set of paths.

Example 2 *Let* $\mathtt{q} : \{\mathtt{Y} \mid \mathtt{Y} \in \mathtt{title}\}$ *be a path query. Then* $\mathtt{imdb} \triangleright \mathtt{q}$ *is a nested path expression. It specifies paths of level* 0 *reduced to a single edge labelled by* `imdb` *and whose destination is a complex node labelled by a nested graph* G_i *on which the query* `q` *evaluates to a non-empty set of paths of level* 1*, reduced to one edge labelled by* `title`*. The single path of the db-graph of figure 1 spelled by* $\mathtt{imdb} \triangleright \mathtt{q}\}$ *is the edge ($n_1, n_2, imdb$). Note that the query q evaluated on the contents of the complex node n_2 returns the 3 edges whose destinations are respectively labelled by* "`Destiny`", "`Psycho`" *and* "`Cosby Show`".

Definition 3 (Nest-Path) *A nested path expression is recursively defined by:*

1. *A path variable or a label term is a nested path expression. In this case, it is both pre-free (the origin of the path is not constrained) and post-free (the destination of the path is not constrained).*

2. *(a)* $s \triangleright t$ *(b)* $t \triangleleft s$
   (c) $s \triangleright q(X_1, \ldots, X_n)$ *(d)* $q(X_1, \ldots, X_n) \triangleleft s$ *are nested path expressions if*

• s is a nested path expression. In cases (b) and (d), s is required to be pree-free and in cases (a) and (c), s is required to be post-free. • t is a data term. • q is a nested path query *of arity n* [1] *• X_i are path variables.*

The nested path expressions of types (a) and (c) are post-bound. Those of types (b) and (d) are pre-bound.

3. *$s_1.s_2$ is a nested path expression when s_1 and s_2 are nested path expressions, resp. post-free and pree-free. $s_1.s_2$ is pre-free (resp. post-free) as soon as s_1 (resp. s_2) is.*

Example 3 *The expression* `movie.title` *captures every path p having two edges labelled successively by* `movie` *and* `title`*. It is pre-free and post-free since neither the destination nor the origin of p is bound by a data term or a query. The path expression* `movie.title` $\triangleright$ "`Destiny`" *captures paths of the same form whose destination brings the data* "`Destiny`"*. This expression is pre-free and post-bound. The expression* `imdb` $\triangleright$ `q(X)` *is a pre-free, post-bound nested path expression.*

We are now going to complete the former definition to make precise what is a path query. A path formula is build from path expressions as follows:

1. It is a nested path expression.
2. It is t_1=t_2 where t_1 and t_2 are terms of the same sort.
3. It is $t \in s$ where t is a path term and s is a nested path expression.
4. $\phi \wedge \psi$ (resp. $\phi \vee \psi$, $\neg\, \phi$, $(\exists \mathbb{X})\ \phi$ and $(\forall \mathbb{X})\ \phi$ where ϕ and ψ are path formulas.

Intuitively, $t \in s$ intends to check that the path t is among the paths spelled by the path expression s.

Definition 4 (Nested path query) *A nested path query of arity n is of the form $\{X_1, \ldots, X_n \mid \varphi\}$ where φ is a nested path formula, for $i = 1..n$, X_i is a path variable, and the set $Free(\varphi)$ of free variables of φ is exactly $\{X_1, \ldots, X_n\}$.*

Example 4 *Let us consider the nested path query* $\mathtt{r} : \{\mathtt{X} \mid \mathtt{imdb} \triangleright \mathtt{q(X)}\}$ *where* `q` *is the unary query* $\{\mathtt{Y} \mid \mathtt{Y} \in \mathtt{title}\}$.

This query is meant to return paths of level 1 embedded in a node which is the destination of an edge of level 0 labelled by `imdb`*. These output paths are reduced to single edges labelled by $title$.*

The preceding example suggests that a level of nesting is associated to variables in path expressions and queries. The notion of level of a variable in a path expression or query is necessary in order to define the semantics. We will restrict our presentation to meaningful examples.

[1]See definition 4.

Example 5 *In the previous example, the level of the variable* $\mathtt{X}$ *occurring in* $\mathtt{r}$ *is* 1. *This is implied by the fact that the level of the variable* $\mathtt{Y}$ *in* $\mathtt{q}$ *is* 0 *and* $\mathtt{X}$ *is linked to* $\mathtt{Y}$ *by* $\mathtt{q}$ *in the nested path expression* $\mathtt{imdb} \triangleright \mathtt{q}(\mathtt{X})$.

The expression $\{\mathtt{X} \mid \mathtt{X.imdb} \triangleright \mathtt{q}(\mathtt{X})\}$ where $\mathtt{q} : \{\mathtt{Y} \mid \mathtt{Y} \in \mathtt{title}\}$ is not a query because the variable $\mathtt{X}$ is assigned two levels (0 and 1) and a concrete path cannot be assigned two different levels. In [17], a procedure has been defined that assigns levels of nesting to the variables of a path query or path expression.

Example 6 *Let us consider once more the db-graph of Figure 1. Assume that the user knows about the "structure" of the information in node* n_2 *as well as the "structure" of the information in node* n_4 *(These structures may have been extracted by an intelligent engine). Thus he/she knows that* n_2 *provides titles together with directors of movies and* n_4 *provides titles together with theaters. Now assume that the user just wants to combine these information to have together titles, directors and theaters. He/she may express this by the following query* $\mathtt{r} : \{\mathtt{F}, \mathtt{D}, \mathtt{T} \mid \mathtt{s}\}$ *where* s *is the following path expression*

$$\mathtt{imdb} \triangleright \mathtt{q}_1(\mathtt{F}, \mathtt{D}) \ \wedge \ (\exists \mathtt{U})(\exists \mathtt{Fm}) \ (\mathtt{U} \triangleright \mathtt{q}_2(\mathtt{Fm}, \mathtt{T}) \wedge (\exists \alpha) \ (\mathtt{F} \in \mathtt{title} \triangleright \alpha \wedge \mathtt{Fm} \in \mathtt{title} \triangleright \alpha))$$

$$\mathtt{q}_1 : \{\mathtt{X}_1, \mathtt{Y}_1 \mid (\exists \mathtt{Z}_1) \ (\mathtt{Z}_1.\mathtt{X}_1 \ \wedge \ \mathtt{X}_1 \in \mathtt{title} \wedge \mathtt{Z}_1.\mathtt{Y}_1 \ \wedge \ \mathtt{Y}_1 \in \mathtt{director})\}$$

$$\mathtt{q}_2 : \{\mathtt{X}_2, \mathtt{Y}_2 \mid (\exists \mathtt{Z}_2)(\mathtt{Z}_2 \in \mathtt{pariscope} \ \wedge \ \mathtt{Z}_2.\mathtt{X}_2 \ \wedge \ \mathtt{X}_2 \in \mathtt{title} \wedge \mathtt{Z}_2.\mathtt{Y}_2 \ \wedge \ \mathtt{Y}_2 \in \mathtt{theater})\}$$

The query $\mathtt{q}_1$ *collects information (title and director) from the complex node* n_2 *and the query* $\mathtt{q}_2$ *returns information (title and theater) from* n_4. *The combination is performed by the condition*

$(\exists \alpha) \ (\mathtt{F} \in \mathtt{title} \triangleright \alpha \ \wedge \ \mathtt{Fm} \in \mathtt{title} \triangleright \alpha)$ *that appears in* $\mathtt{r}$. *This combination acts like a join.*

Now that we have a way to select paths (and embedded paths) in a db-graph, we naturally can select sub-graphs (and embedded sub-graphs) via the language Graph(Nest-Path). Let us reuse the preceding example to illustrate this language.

3.3. Emb-Path: a fixpoint nested path calculus

In this section, we introduce a new path language called *Emb-Path* to query nested semi-structured data without knowing, in advance, the level where data are situated.

3.4. Emb-Path's syntax and semantics

Roughly speaking, an embedded path formula is specified by two Path formulas. The evaluation of the embedded path formula specified by φ and ψ is an iterative process. At each step, the role of the first formula φ is simply to extract paths of interest. At each step, the second formula ψ serves to access some specific complex nodes in the db-graph, the ones which are going to be unnested in the sense that the labels of these complex nodes which are db-graphs are going to be processed (queried) at the next iteration. Thus in fact the second formula allows one to navigate in depth into the db-graph (not necessarily in the whole db-graph). Formally,

Definition 5 (Emb(Path,Path)) *Let φ be a Path formula with n free path variables $X_1, \ldots, X_n$ and let ψ be a Path formula with one free path variable Y. Then $Emb(\varphi, \psi)$ (read φ is embedded in ψ) is an embedded path expression of arity n. Given a db-graph G, $Emb(\varphi, \psi)$ denotes the n-ary relation which is the limit of the sequence $\{x_k\}_{k \geq 0}$ defined by:*

1. *x_0 is the answer to the query $\{(X_1, \ldots, X_n) \mid \varphi\}$ evaluated on G.*
   y_0 is the answer to the query $\{(Y) \mid \psi\}$ evaluated on G.
2. *$x_{k+1} = x_k \cup \varphi(y_k)$ and $y_{k+1} = y_k \cup \psi(y_k)$ where*

   $\varphi(y_k)$ (resp. $\psi(y_k)$) is the union of the answers to the query $\{(X_1, \ldots, X_n) \mid \varphi\}$ (resp. $\{(Y) \mid \psi\}$) evaluated on the db-graphs labelling the destination of the paths $p \in y_k$ (when this destination is a complex node).

An atomic embedded formula is an expression of the form $Emb(\varphi, \psi)(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are path terms and an embedded path expression $Emb(\varphi, \psi)$ of arity n. Note that a Path formula φ having n free variables can be "translated" by embedding φ in false. The language of embedded formulas is denoted Emb-Path as an abbreviation of Emb(Path,Path). It leads to a graph query language in a straightforward manner by considering Graph(Emb-Path).

Note that the above definition can be generalized in a standard way by allowing φ and ψ to be embedded formulas themselves. In this case, the language of embedded formulas is called Emb(Emb-Path,Emb-Path).

Example 7 *Let us consider the introductory query example: the user is trying to gather all movie or tv-serie titles in the database no matter at which level of nesting they are. In order to write this query, we will use the Emb-Path expression* $\mathtt{Emb}(\varphi, \psi)$ *of arity one where:* φ *is* $\mathtt{U} \in \mathtt{title}$ *and* ψ *is* $\mathtt{V}$
The graph query which returns all the titles is written in Graph(Emb-Path) as:

$\{ \mathtt{X} \mid \exists \mathtt{Y}\, \mathtt{Emb}(\varphi, \psi)(\mathtt{Y}) \wedge \mathtt{Y}[\mathtt{X}] \}$

This graph query just extracts the subgraphs rooted at the destination of the paths Y satisfying the embedded formula Emb(φ, ψ)(Y). What kind of paths does the embedded formula select? Because the path formula φ is $\mathtt{U} \in \mathtt{title}$, *Emb($\varphi$, ψ) returns embedded paths reduced to edges labelled by* `title`. *Because the path formula ψ is* `V`, *all complex nodes are explored and unnested and thus Emb(φ, ψ) returns all paths reduced to edges labelled by* `title` *and belonging to any level of nesting.*

Definition 6 (Emb(Graph,Path)) *Let φ be a Graph(Path) formula with n free graph variables X_1, $\ldots$, X_n and let ψ be a Path formula with one free path variable Y. Then $Emb(\varphi, \psi)$ (read φ is embedded in ψ) is an embedded graph expression of arity n. An atomic formula of Emb(Graph,Path) is an expression of the form $Emb(\varphi, \psi)(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are graph terms.*

Example 8 *The query* $\{ \mathtt{X} \mid \mathtt{Emb}(\varphi, \psi)(\mathtt{X}) \}$ *where* φ *is* $\mathtt{title}[\mathtt{Y}]$ *and* ψ *is* `V` *return the graph containing information about title at any level of nesting.*

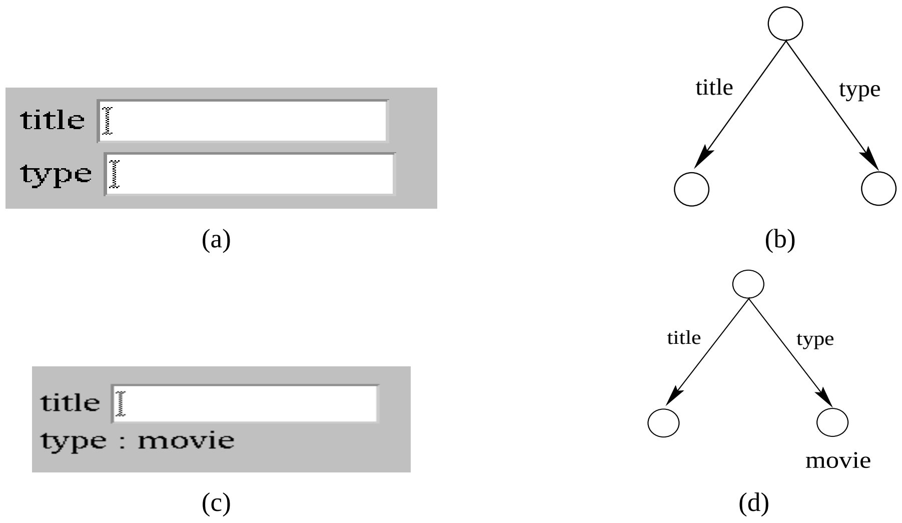

Figure 2. Two forms with their schemas.

Example 9 *Let us consider the following query : "Who are the directors of the film* `Destiny` *and its schedule in cinema* `Gaumont`*?". This query can be expressed in Emb(Graph,Path) as follows:*

$$\mathtt{q} : \{(\mathtt{D}, \mathtt{S}) \mid \mathtt{Emb}(\varphi_1, \mathtt{V})(\mathtt{D}) \ \wedge\ \mathtt{Emb}(\varphi_2, \mathtt{V})(\mathtt{S})\}$$

φ_1 *est* $(\exists \mathtt{Y}_1)\ (\mathtt{Y}_1 : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge\ \mathtt{Y}_1 : \mathtt{type} \triangleright \mathtt{movie} \ \wedge\ \mathtt{Y}_1 : \mathtt{director}[\mathtt{X}_1])$

φ_2 *est* $(\exists \mathtt{Y}_2)\ (\mathtt{Y}_2 : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge\ \mathtt{Y}_2 : \mathtt{theater} \triangleright \text{“}\mathtt{Gaumont}\text{”}$
$\wedge\ \mathtt{Y}_2 : \mathtt{schedule}[\mathtt{X}_2])$

4. Forms modelling

As we have mentioned in the introduction, a Web form, like database view, is used to limit the type of queries that can be posed on the Web. The problem addressed, in this section, is how to traduce a query posed on semi-structured data and web forms to a set of queries posed on the nested db-graph that represents the semi-structured data as well as the web forms. In the next paragraphs, we will describe how to model the Web in the presence of forms.

The Web, as we have seen, can be modelled by a (nested) db-graph however a form is associated to a parameterized query which defines its capacity, i.e., what a form should return to user. Let $\mathrm{Graph}^{\mathrm{Conj}}(\mathcal{L}_{path})$ or simply $\mathrm{Graph}^{\mathrm{Conj}}$ be a sub-language of Graph($\mathcal{L}_{path}$) (see definition 2). The sub-language $\mathrm{Graph}^{\mathrm{Conj}}$ is constituted from conjunctive queries without negation nor equality or bisimulation symbols. When $\mathrm{Graph}^{\mathrm{Conj}}$ is restricted to atoms of the form $t : s$, it is called $\mathrm{Graph}^{\mathrm{RConj}}$.

A Form schema is simply a db-graph representation of the form. For example the db-graph of figure 2.b is a schema of the form of figure 2.a. The instance of this schema is the db-graph of node n_2 in figure 1. In the next paragraph, we will define formally a parameterized query from $\mathrm{Graph}^{\mathrm{RConj}}$ that will be used to describe a form schema and its capacity (i.e, the set of answers that can be returned by a form).

Definition 7 [Paremeterized Query] *Let $S = (N, E, \lambda, org, dest)$ be a form schema where card(E)=n. A parameterized query associated with S is a query in the language $\mathrm{Graph}^{\mathrm{RConj}}$ of the form: $\{U \mid a_1 \wedge \ldots \wedge a_n\}$ such that:*

$$a_i = \begin{cases} U : \lambda(e_i) \rhd \lambda(dest(e_i)) & \textit{if } \lambda(dest(e_i)) \textit{ is defined} \\ U : \lambda(e_i) \rhd \alpha_i & \textit{otherwise} \end{cases}$$

The variables $\alpha_1, \ldots, \alpha_m$ are called parameters.

Example 10 *The parameterized query $\{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \rhd \alpha \ \wedge \ \mathtt{U} : \mathtt{type} \rhd \beta\}$ with parameters α et β, is associated to the form schema of figure 2.b. However the form schema of figure 2.d is modelled by the parameterized query $\{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \rhd \alpha \ \wedge \ \mathtt{U} : \mathtt{type} \rhd \mathtt{movie}\}$.*

5. Querying semi-structured data and Forms

Recall that the Web is modelled by a (nested) db-graph however the forms are modelled by parameterized queries described their schemas. The user wants to query the web, in non nested languages, without making attention to the presence or absence of the forms. The query of example 9 is reformulated in the language $\mathrm{Graph}^{\mathrm{Conj}}$ like this:

$$\begin{aligned} \mathtt{q} : \{(\mathtt{D}, \mathtt{S}) \mid \ & (\exists \mathtt{Z}_1)\,(\mathtt{Z}_1 : \mathtt{title} \rhd \text{“}\mathtt{Destiny}\text{”} \ \wedge \ \mathtt{Z}_1 : \mathtt{type} \rhd \mathtt{movie}) \\ & \wedge\, (\exists \mathtt{Z}_2)\,(\mathtt{Z}_2 : \mathtt{theater} \rhd \text{“}\mathtt{Gaumont}\text{”} \ \wedge \ \mathtt{Z}_2 : \mathtt{title} \rhd \text{“}\mathtt{Destiny}\text{”}) \\ & \wedge\, \mathtt{Z}_1 : \mathtt{director}[\mathtt{D}] \ \wedge \ \mathtt{Z}_2 : \mathtt{schedule}[\mathtt{S}]\} \end{aligned}$$

Although the query q is non nested, but we consider that its “intention” is to return data from different levels of nesting by taking into account the presence of forms modelled by parameterized queries. Hence, the query q will be translated in our framework into many queries from Emb(Graph,Path). The notion of translation is given bellow:

Definition 8 [Translation] *Let q be a query from $\mathrm{Graph}^{\mathrm{Conj}}$ and let G be nested db-graph modelling the Web. $\mathbb{G}$ is the union of db-graphes nested in G and $\mathbb{V}$ is the set of parameterized queries associated with web forms. The translation of the query q consists to find a set of queries Out from Emb(Graph,Path) such that:*

1. *for each q' from Out, $q'(G) \subseteq q(\mathbb{G})$ and*
2. *for each query q' from Out, there is no query q'' from Out verifying*

$$q'(G) \subset q''(G) \subseteq q(\mathbb{G}).$$

In the rest of the presentation we need the notion of subsumption between queries of $\mathrm{Graph}^{\mathrm{RConj}}$. A lot of subsumption tests were proposed in the context of conjunctive relational queries [15]. We adopt here a test based on the existence of an inclusion mapping [7].

In the remainder, $exp(\varphi)$ denotes the set of sub-path expressions appeared in a formula φ. For example, for φ equals $a \ \wedge \ X : b.Y$ we obtain $exp(\varphi) = \{a, b, Y, b.Y\}$.

Definition 9 [Inclusion mapping] *Let q and r be two queries of $\mathrm{Graph}^{\mathrm{Conj}}$ of the form $\{U \mid \varphi\}$ and $\{X \mid \phi\}$. Let $var(\varphi)$ (resp. $var(\phi)$) be the set of variables of φ (resp. ϕ). Let $adom(\varphi)$ be the active domain of φ (i.e., set of constants appearing in φ). Let $exp(\varphi)$ be the set of sub-path expressions appeared in a formula φ. Let θ be a mapping from $var(\phi)$ to $var(\varphi) \cup adom(\varphi) \cup exp(\varphi) \cup \{\epsilon\}$.*

1. *θ is an inclusion mapping from ϕ to φ if for each atom a_ϕ from ϕ, there exists an atom a_φ from φ such that $\theta a_\phi = a_\varphi$* [2].
2. *θ is an inclusion mapping from r to q if*
   (a) *$U = \theta(X)$ and*
   (b) *θ is an inclusion mapping from ϕ to φ.*

Example 11 *Let us consider the following three cases of the queries* `q` *and* `r`*:*

1. $\mathtt{q} : \{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge\ \mathtt{U} : \mathtt{type} \triangleright \mathtt{movie}\}$
   $\mathtt{r} : \{\mathtt{X} \mid \mathtt{X} : \mathtt{title} \triangleright \alpha \ \wedge\ \mathtt{X} : \mathtt{type} \triangleright \beta\}$
2. $\mathtt{q} : \{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \triangleright \alpha\}$
   $\mathtt{r} : \{\mathtt{X} \mid (\exists \mathtt{V})\ \mathtt{X} : \mathtt{V}\}$
3. $\mathtt{q} : \{\mathtt{U} \mid \mathtt{U} : \mathtt{type} \triangleright \mathtt{movie}\}$
   $\mathtt{r} : \{\mathtt{X} \mid (\exists \mathtt{V})\ \mathtt{X} : \mathtt{type}.\mathtt{V} \triangleright \mathtt{movie}\}$

The three inclusion mappings θ_i from `r` *to* `q` *are given bellow:*

1. $\theta_1 : \mathtt{X} \longrightarrow \mathtt{U}, \alpha \longrightarrow \mathtt{Destiny}, \beta \longrightarrow \mathtt{movie}$
2. $\theta_2 : \mathtt{X} \longrightarrow \mathtt{U}, \mathtt{V} \longrightarrow \mathtt{title} \triangleright \alpha$
3. $\theta_3 : \mathtt{X} \longrightarrow \mathtt{U}, \mathtt{V} \longrightarrow \epsilon$

Lemma 1 *Let q and r be two restricted conjunctive queries from $\mathrm{Graph}^{\mathrm{RConj}}$. $q \subseteq r$ if and only if there exists an inclusion mapping from r to q.*

In figure 3 we propose an algorithm that can be used to translate a query of one free graph variable to a set of embedded queries. The case of many free graph variables is similar. The procedure $Combine(\{X \mid \varphi\}, Out)$ replaces each atom a_i (different from Emb) of φ, where Free(a_i)=$\{\mathbb{X}_1, \ldots, \mathbb{X}_n\}$, by $Emb(a_i, V)(\mathbb{X}_1, \ldots, \mathbb{X}_n)$ and then add the resultant query to the set Out. Although the embedding definition (see definition 5) allows only free path variable, we can easily bypass this limit to allow also data or label variables to appear free. In the algorithm, the set L is used to store semi-translated queries. The lines 10..16 use a form described by a parameterized query to match a query from L. The lines 18..22 use a portion of a query from L to instantiate parameters of a parameterized query associated with a form.

[2] θa_ϕ is the atom a_ϕ where each variable $\mathbb{X}$ of $var(a_\phi)$ is replaced by $\theta(\mathbb{X})$. The extension of θ for a formula is straightforward.

```
Require:  $q$ a query from $\mathrm{Graph}^{\mathrm{Conj}}$ of the form $\{X_1 \mid \varphi_1\}$.
          $\mathbb{V}$ a set of parameterized queries associated to the schemas of forms.
Ensure:   $Out$ a set of the translations of $q$
 1: $Out \leftarrow \emptyset$;
 2: $L \leftarrow \emptyset$;
 3: $Combine(\{X_1 \mid \varphi_1\}, Out)$;
 4: insert $q$ in $L$;
 5: while $L$ is non empty do
 6:   extract a query $\{X \mid \varphi\}$ from $L$;
 7:   $\varphi$ is set in a prenex form $(\exists \mathbb{X}_1) \ldots (\exists \mathbb{X}_n)\ \varphi_2$;
 8:   for each query $r : \{U \mid \phi(U, \alpha_1, \ldots, \alpha_m)\}$ of $\mathbb{V}$ do
 9:     for each sub-formula $\psi$ from $\varphi_2$ do
10:       if $r$ is without parameters then
11:         if exists an inclusion mapping $\theta$ from $\psi$ to $\phi$                                  then
             and $\theta$ is not an inclusion mapping to $\phi$ from any other sub-formula of $\varphi_2$ except $\psi$
12:           $\xi$ is resulted from $\varphi$ by replacing $\psi$ by $Emb(\phi, V)(U)$;
13:           $\xi \leftarrow \theta\xi$;
14:           insert $\{Z \mid \xi(Z)\}$ in $L$;
15:           $Combine(\{Z \mid \xi(Z)\}, Out)$;
16:         end if
17:       else
18:         if exists an inclusion mapping $\theta$ from $\phi$ to $\psi(Y)$                             then
             and $\theta$ is not an inclusion mapping from $\phi$ to any other sub-formula of $\varphi_2$ except $\psi$
             and for all $\alpha_i$, $\theta(\alpha_i)$ is a constant
19:           $\xi$ is resulted from $\varphi$ by replacing $\psi$ by $Emb(\theta\phi, V)(Y)$;
20:           insert $\{X \mid \xi\}$ in $L$;
21:           $Combine(\{X \mid \xi\}, Out)$;
22:         end if
23:       end if
24:     end for
25:   end for
26: end while
27: return $Out$.
```

Figure 3. Query translation.

Example 12 *Let* $\mathtt{r}$ *be a parameterized query associated with the form schema (figure 2.b) of the the form in figure 2.a. Remember that the instance of this schema is the db-graph of node* n_2 *in figure 1.*

$$\mathtt{r} : \{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \triangleright \alpha \ \wedge \ \mathtt{U} : \mathtt{type} \triangleright \beta\}$$

The following query $\mathtt{q}$ *from* $\mathrm{Graph}^{\mathrm{Conj}}$ *returns all directors of the film* "$\mathtt{Destiny}$".

$$\mathtt{q} : \{\mathtt{X} \mid (\exists \mathtt{N})\ (\mathtt{N} : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge \ \mathtt{N} : \mathtt{type} \triangleright \mathtt{movie} \ \wedge \ \mathtt{N} : \mathtt{director}[\mathtt{X}])$$

The set of translations Out *obtained from the algorithm of figure 3 contains the two following embedded queries:*

1. $\{\mathtt{X} \mid (\exists \mathtt{N})\ (\mathtt{Emb}(\mathtt{N} : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”}, \mathtt{V})(\mathtt{N})$
$\wedge\, \mathtt{Emb}(\mathtt{N} : \mathtt{type} \triangleright \mathtt{movie}, \mathtt{V})(\mathtt{N})$
$\wedge\, \mathtt{Emb}(\mathtt{N} : \mathtt{director}[\mathtt{X}], \mathtt{V})(\mathtt{N}, \mathtt{X}))\}$

2. $\{\mathtt{X} \mid (\exists \mathtt{N})\ (\mathtt{Emb}(\mathtt{N} : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge \ \mathtt{N} : \mathtt{type} \triangleright \mathtt{movie}, \mathtt{V})(\mathtt{N})$
$\wedge\, \mathtt{Emb}(\mathtt{N} : \mathtt{director}[\mathtt{X}], \mathtt{V})(\mathtt{N}, \mathtt{X}))\}$

The first query explores the Web by querying all nested db-graphs of figure 1. The first embedding of the second query queries the db-graph of level 1 and the db-graph situated in node n_2*. The parameter* α *and* β *of the parameterized query are instantiated respectively by* "$\mathtt{Destiny}$" *and* $\mathtt{movie}$*. The first and the second embeddings of the second query are both posed on the same db-graph because they share the same free graph variable* $\mathtt{N}$*. Some optimizations techniques can be proposed in such case, instead of generating two embeddings we can generate only one embedong.*

Example 13 *The two parameterized queries* $\mathtt{r}_1$ *and* $\mathtt{r}_2$ *given bellow describe schemas of two forms. The first form has two attributes* `title` *and* `type`*. The second form has also two attributes* `theater` *and* `title`.

$$\mathtt{r}_1 : \{\mathtt{U} \mid \mathtt{U} : \mathtt{title} \triangleright \alpha \ \wedge \ \mathtt{U} : \mathtt{type} \triangleright \beta\}$$

$$\mathtt{r}_2 : \{\mathtt{U} \mid \mathtt{U} : \mathtt{theater} \triangleright \alpha \ \wedge \ \mathtt{U} : \mathtt{title} \triangleright \beta\}$$

Let us $\mathtt{q}$ *be the following query :"Who are the directors of the film* `Destiny` *and its schedule in cinema* `Gaumont`*?" (see example 9).*

$$\begin{aligned} \mathtt{q} : \{(\mathtt{D}, \mathtt{S}) \mid\ & (\exists \mathtt{N}_1)\ (\mathtt{N}_1 : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”} \ \wedge \ \mathtt{N}_1 : \mathtt{type} \triangleright \mathtt{movie}) \\ & \wedge (\exists \mathtt{N}_2)\ (\mathtt{N}_2 : \mathtt{theater} \triangleright \text{“}\mathtt{Gaumont}\text{”} \ \wedge \ \mathtt{N}_2 : \mathtt{title} \triangleright \text{“}\mathtt{Destiny}\text{”}) \\ & \wedge \mathtt{N}_1 : \mathtt{director}[\mathtt{D}] \ \wedge \ \mathtt{N}_2 : \mathtt{schedule}[\mathtt{S}]\} \end{aligned}$$

The set of translations Out *obtained from the algorithm of figure 3 can be given straightforwardly.*

6. Conclusion

In this paper we have shown how to model and query nested semi-structured data in the presence of the Web forms. The formalism used here is based on logic. The syntax of our languages can be modified to be near the syntax of XML query languages: XPath and XQuery [16]. The open problem that can be studied deeply is how to query nested semi-structured data by example or by forms like it is done in [8].

References

[1] Serge Abiteboul. Querying semistructured Data. In *ICDT*, pages 1–18, 1997

[2] S. Abiteboul and N. Bidoit. Non first normal form relations: An algebra allowing data restructuring. *Journal of Computer and System Sciences, Academic*, 33, 1986.

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

[4] Nicole Bidoit and Mourad Ykhlef. Fixpoint Path Queries. In *International Workshop on the Web and Databases WebDB'98 In Conjunction with EDBT'98*, pages 56–62, Valencia, Spain, 27–28 March 1998.

[5] Nicole Bidoit and Mourad Ykhlef. Fixpoint Calculus for Querying Semistructured Data. *Lecture Notes in Computer Science*, 1590:78–98, 1999.

[6] Nicole Bidoit, Sofian Maabout and Mourad Ykhlef. A familly of of nested query languages for semistructured data. *Lecture Notes in Computer Science*, 1762:13–30, 2000.

[7] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. Proc. 9th Ann. ACM Symposium on the Theory of Computing. 77–90,1977

[8] Irna M. R. Evangelista Filha and Alberto H. F. Laender and Altigran Soares da Silva. Querying Semistructured Data By Example: The QSByE Interface. Workshop on Information Integration on the Web, pages 156–163,2001

[9] Hector Garcia-Molina and Wilburt Labio and Ramana Yerneni, ICapability-Sensitive Query Processing on Internet Sources. IICDE, pages 50–59, 1999

[10] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. VLDB pages 54–65, Sep 1995.

[11] David Konopnicki and Oded Shmueli. Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QSSystem. ACM Trans. Database Syst. Volume 23, number 4, pages 369-410, 1998,

[12] Alberto H. F et al. Representing Web Data as Complex Objects, EC-Web, pages:216–228, 2000

[13] R. Milner. Communication and concurrency. Prentice Hall, 1989.

[14] A. Silva and I. Filha and A. Laender and D. Embley. Representing and querying semistructured web data using nested tables with structural variants. In ER, 2002

[15] J. D. Ullman. Information Integration Using Logical Views, *icdt*, 19–40, 1997.

[16] http://www.w3.org/XML/Query

[17] M. Ykhlef. *Interrogation des données semistructurées*. PhD thesis, Univ. of Bordeaux, France 1999.