Interrogation de Données Semi structurées sur Internet

Ykhlef Mourad
Foc@lNet Bat C101, 27 avenue de Galliéni
92160 Antony (France)
E-mail : ykhlef@yahoo.fr

1 Introduction

Pendant longtemps, les bases de données respectant certaines structurations, ont été les sources essentielles d'information. Faire communiquer ou coopérer ces sources revenaient à faire de l'intégration de bases de données. Maintenant que les bases de données n'ont plus ce monopole, surtout depuis l'émergence de l'Internet (Web), l' intégration se retrouve plus compliquée puisqu'on ne peut plus compter sur la forte structuration des bases de données. Ainsi la notion de données semistructurées (HTML, XML . . .) à vu le jour.

Le Web s'est développé pour permettre l'accès à des fichiers chaînés (hypertexte). Rapidement, la nécessité de couplage avec les bases de données est apparue afin de permettre la génération de sites Web dynamiques composés à partir de « templates » HTML et d'informations extraites de bases de données. Les applications en commerce électronique utilisent souvent des bases de données puisqu'il s'agit d'accéder, à travers le Web, à des catalogues.

Le semistructuré parait bien adapté pour le Web. Il permet la prise en compte de documents HTML/XML et facilite l'interrogation et l'intégration de données provenant de plusieurs sources hétérogènes. Industriels et académiques se sont largement penchés sur les données semistructurées pour répondre aux exigences du monde Web.

Plusieurs travaux ont porté sur la modélisation et l'interrogation [QRS + 95, BDS95, BDHS96, FFLS97, AM98] [BY98, BY99, BMY99, BMY00] des données semistructurées. Ces dernières sont généralement modélisées par des graphes orientés et étiquetés. Les langages d'interrogation de données semistructurées proposés dans la littérature expriment au mieux des requêtes qui sont dans la classe de requêtes du premier ordre plus la fermeture transitive FO+TC, lorsque les graphes représentent des relations. Le langage de requêtes du type calcul Graph-Fixpoint [BY98,

BY99, Ykh99] pour interroger des données semistructurées modélisées par des db-graphes ex-prime des requêtes dans la classe Fixpoint [AHV95]. Il est donc strictement plus puissant que les autres langages existants.

Nous revoyons, dans cet article, les différents modèles de données semistructurées proposés dans la littérature et les langages d'interrogation. La prochaine section est consacrée à la présentation des données semistructurées. La troisième et la quatrième sections sont consacrées à la modélisation et aux langages d'interrogation des données semistructurées. Dans la cinquième section on montre l'utilisation du modèle (resp. langage) relationnel pour la modélisation (l'interrogation) des données semistructurées. La dernière section présente les expressions de chemin qui sont au coeur des langages d'interrogation des données semistructurées.

2 Les données semistructurées

L'adjectif « semistructuré » est apparu pour contraster les données provenant de bases de données relationnelles [Cod70] et les bases de données orientées objets qui sont fortement structurées. On peut être en présence de données semistructurées dès qu'on intègre deux bases de données. En effet, supposons que les deux contiennent des informations sur des films. Dans l'une nous avons en plus du titre, le nom des acteurs et l'heure de projection en type Date et dans l'autre, nous n'avons pas le nom des acteurs mais plutôt le nom du metteur en scène et l'heure de projection est de type Chaîne de caractères. On se retrouve donc avec des données qui n'ont pas une structure régulière.

On peut aussi parler de données semistructurées dans le contexte du Web où les données se présentent sous forme de fichiers HTML . Pour interroger les documents HTML on a besoin d'extraire la structure non régulière des données qui y résident. Dans la plupart des cas cette extraction reste un défi difficilement surmontable. En pratique, l'interrogation du Web est faite par des moteurs de recherche à base de mots clés. A½n de résoudre ce problème, on a proposé le standard XML [BPSM98] qui rend explicite la structure, même non régulière, des données.

3 Modélisation des données semistructurées

Les données semistructurées sont généralement modélisées par des graphes orientés et étiquetés.
Trois variétés de graphes ont été proposées : graphe non imbriqué, h-arbre et db-graphe imbriqué.

3.1 Modèle graphe non imbriqué

Nous allons présenter le modèle OEM (pour Object Exchange Model [PGW95]), le modèle Arbre UP [BDHS96] et le modèle db-graphe [BY98, BY99].

3.1.1 Le modèle OEM

Le modèle OEM proposé dans le cadre du projet Tsimmis [CGMH + 94], est inspiré du modèle orienté objet ODMG [Cat97]. OEM permet de modéliser les données semistructurées sous forme de graphe où les nœuds sont associés aux objets et les arcs portent des étiquettes décrivant les liaisons qui existent entre les objets. La figure 1 donne un exemple d'un graphe OEM modélisant les films et les feuilletons. Les identificateurs de nœuds sont placés à

l'intérieur des nœuds (n_i identifie le i^e nœud). Les nœuds feuilles sont associés aux objets simples portant des valeurs simples (Destiny Chahine 1 . . .) alors que les nœuds internes sont associés aux objets complexes.
Remarquons que la structure de ce graphe est irrégulière, l'information sur le genre du ½lm associé au nœud n_5 est absente par contre le genre du film n_6 est connu (drama). La représentation

Figure 1: Un graphe OEM modélisant des données semistructurées.

textuelle du graphe OEM de la figure 1 est
&n_1 { motionpicture :&n_2
{ movie :&n_5
{ title : &n_8 Destiny,
director :&n_9 Chahine } } ,
motionpicture :&n_3
{ movie :&n_6
{ title :&n_{10} Apocalypse Now,
director :&n_{11}.Coppola
genre :&n_{12} Drama } } ,
motionpicture :&n_4
{ tvserie :&n_7
{ title :&n_{13} Cosby Show
director :& n_{14} Sandrich
director :& n_{15} Singletary
episode :& n_{16} 1
episode :& n_{17}100 }}}

3.1.2 Le modèle Arbre UP

Le modèle Arbre UP modélise les données semistructurées par un arbre où les étiquettes sont toujours sur les arcs contrairement au modèle OEM où les étiquettes peuvent être associées aux nœuds terminaux. Un Arbre UP est défini récursivement comme suit :
1. { } est un Arbre UP vide,
2. { l : t } est un Arbre UP construit à partir d'un Arbre UP t et un arc étiqueté par l pointant vers la racine de t[1]

[1] { l : {}} est réécrit en { l } .

Figure 2: Un Arbre UP.

Figure 3: Deux graphes égaux dans Arbre UP et non égaux dans OEM.

3. $t_1 U\ t_2$ (ou { t_1 t_2}) est un Arbre UP obtenu en fusionnant les racines de t_1 et t_2.

La figure 2 est une représentation graphique de l'Arbre UP suivant :

```
{ motionpicture : { movie :    { title : Destiny
                                 director : Chahine } } ,
motionpicture : { movie :      { title : ApocalypseNow
                                 director : Coppola
                                 genre : drama } } ,
motionpicture : { tvserie :    { title : CosbyShow
                                 director : Sandrich
                                 director : Singletary
                                 episode : 1
                                 épisode : 100 } } }
```

Le modèle Arbre UP n'utilise pas la notion d'identificateur d'objet comme c'est le cas pour OEM. Ainsi, il ne peut pas exprimer le partage d'arbres. Les deux graphes de la figure 3, qui ne sont pas égaux par OEM, sont égaux du point de vue du modèle Arbre UP. Deux Arbres UP

Figure 4: Modélisation par h-arbre du Web.

sont égaux si et seulement s'ils sont bisimilaires [Mil89]. La bisimulation est comparable à la notion d'égalité par valeur entre objets et elle est calculable en temps polynômial.

3.1.3 Le modèle db-graphe

Le db-graphe est proposé dans [BY98, BY99]. Il se distingue de OEM par la possibilité d'associer des valeurs atomiques aux nœuds non terminaux. Un db-graphe permet de représenter les données semistructurées plus “fidèlement” que OEM et Arbre UP. Par exemple, le Web est modélisé par un db-graphe où chaque document HTML est modélisé par un nœud étiqueté par le contenu (texte) de ce document et les liens hypertextes entre les documents sont modélisés par des arcs étiquetés. OEM modélise le Web en ajoutant, pour chaque document avec des liens hypertextes, un arc auxiliaire pointant vers un nœud simple portant le texte du document.

3.1.4 Discussion

Les modèles OEM, Arbre UP et db-graphe permettent de modéliser les données semistructurées par des graphes non imbriqués. Ils ne sont pas adéquats pour modéliser les données semistructurées dans le cas où ces dernières sont imbriquées. Par exemple, pour une page HTML contenant un tableau, il est souhaitable de la modéliser par un nœud (complexe) étiqueté par un graphe modélisant la structure tableau. Pour pouvoir tenir compte de la nature imbriquée des données semistructurées, deux extensions du modèle graphe ont été proposées. Le modèle h-arbre [AM98] et le modèle db-graphe imbriqué [BMY99, BMY00].

3.2 Le modèle h-arbre

Le modèle h-arbre a été proposé pour modéliser le Web et les structures des documents HTML. Un h-arbre est un arbre ordonné (les sous-arbres sont ordonnés) avec deux types d'arc : arc interne et arc externe. Un arc interne est étiqueté par un enregistrement modélisant la structure

Figure 5: Un h-arbre modélisant une structure d'une page HTML.

d'un document HTML alors qu'un arc externe est étiqueté par un enregistrement décrivant un lien hypertexte qui existe vers un autre document HTML. La figure 4 présente un graphe modélisant le Web. Les h-arbres situés dans les cadres pointillés modélisent les structures de pages HTML. Les arcs pointillés sont des liens externes. Le Web peut avoir une page spéciale appelée schéma qui fournit des points d'entrées pour naviguer sur le Web (ce sont les bookmarks). Si le schéma est absent, il faut l'adresse url (par exemple, http : / / one) d'un document pour y accéder et récupérer les données. Si on considère que le graphe de la figure 1 est une structure possible d'un document HTML alors l'h-arbre correspondant est donné dans la figure 5. Cet h-arbre a un seul arc externe modélisant un lien hypertexte vers un autre document. Cet arc externe est étiqueté par l'enregistrement [label : picture url : http : / / p.gif] fournissant des informations sur l'étiquette et l'adresse url du document référencé.

3.2.1 Discussion

Le fait de ne pouvoir étiqueter les arcs que par des enregistrements est trop restrictif. Prenons par exemple le feuilleton CosbyShow de la figure 1 dont les données ne sont pas représentables par un enregistrement. Deux solutions sont possibles : (i) représenter le feuilleton en question par cinq arcs internes qui ont la même origine et étiquetés respectivement par [title : CosbyShow] , [director : Sandrich] , [director : Singletary] , [episode : 1] et [episode : 100] . Cette solution ne préserve pas le lien sémantique entre les attributs d'un feuilleton; (ii) dupliquer les informations comme dans la figure 5. Nous pensons qu'il est préférable d'étendre le modèle h-arbre en permettant l'utilisation de h-arbre comme étiquettes d'arcs.

Bien que le modèle h-arbre permet de représenter la structure d'un document HTML et les liens entre les documents du Web, nous pensons qu'il est préférable d'utiliser un modèle à base de graphe imbriqué qui permet de mieux rendre compte de l'origine des données et donc préserve d'avantage la sémantique. Dans la section suivante un modèle est présenté pour tenir compte de la nature imbriquée des données semistructurées et repousser les limites du modèle h-arbre.

3.3 Le modèle db-graphe imbriqué

Un db-graphe imbriqué [BMY99, BMY00] est une généralisation du modèle db-graphe présenté dans la section 3.1.3 dans le sens où un nœud peut contenir un graphe. Le modèle db-graphe imbriqué permet de modéliser fidèlement les données semistructurées parce qu'il tient compte de la nature imbriquée de celles ci. Par exemple, le db-graphe imbriqué de la figure 6 modélise une

Figure 6: Modélisation par db-graphe imbriqué du Web.

portion possible du Web. Les identificateurs de nœuds n_1, n_2, . . . sont placés près des nœuds. Les données qui devraient être placées à l'intérieur des nœuds sont elles aussi placées près des nœuds correspondants pour ne pas encombrer la figure. Ce db-graphe contient deux nœuds complexes n_2 et n_4. Le nœud n_2 destination de l'arc portant l'étiquette **imdb** contient un db-graphe imbriqué fournissant des informations sur des films et les feuilletons. Le noeud n_4 est étiqueté par un db-graphe imbriqué qui est une instance de la base de données relationnelle ayant pour schéma pariscope où le schéma de la relation pariscope est pariscope title, schedule). Le nœuds simple n_3 est une racine d'un graphe fournissant des informations sur des films. Un document HTML avec une structure interne est représenté par un nœuds étiqueté par un graphe modélisant la structure du document.

4 Interrogation des données semistructurées

Dans la suite de la présentation l'expression graphe enraciné en n désigne un graphe constitué de tous les arcs accessibles à partir de n .

4.1 Le langage Lorel

Lorel [QRS + 95, AQM + 97] (pour Lightweight Object REpository Language) est un langage d'interrogation de données semistructurées modélisées par des graphes OEM. Lorel est inspiré

Figure 7: Représentation d'une base de données relationnelle dans OEM.

du langage orienté objet OQL. La requête Lorel ci-dessous posée sur le graphe de la figure 1, retourne les directeurs du film Destiny

```
select  X director
from    motionpicture movie X
where   X title =“ Destiny”
```

L'évaluation de cette requête correspond à : (1) trouver tous les chemins dans la base réalisant l'expression de chemin **motionpicture movie** qui sont donc constitués de deux arcs étiquetés consécutivement par **motionpicture** et **movie,** puis valuer la variable X par les graphes destinations de ces chemins. Ainsi dans l'exemple, à X sont associés successivement les deux graphes enracinés en n_5 et n_6. (2) Retenir seulement les graphes X dont la racine est l'origine d'un arc étiqueté par title et la destination de cet arc porte la valeur Destiny (X. title = “Destiny”).Dans l'exemple on garde le graphe enraciné en n_5. (3) Retourner les graphes accessibles par un arc étiqueté par director (X director). Ainsi, le graphe retourné est celui enraciné en n_8.

Bien que la syntaxe de Lorel ressemble à celle de OQL, ces deux langages sont très différents du point de vue sémantique ([QRS + 95] donne une comparaison détaillée entre les deux langages). La différence la plus significative est que les expressions de chemin utilisées par Lorel étendent les expressions de chemin de OQL (composition d'accès élémentaires à des étiquettes d'arcs) par l'ajout de variables qui peuvent être de trois types : (i) variables de donnée, (ii) d'étiquette ou (iii) de chemin. Ceci permet d'assouplir le mécanisme d'interrogation : une connaissance du schéma n'est plus requise.

Lorel n'est pas assez puissant pour exprimer la fermeture transitive d'une relation binaire représentée sous forme d'un graphe OEM (voir figure 7). Lorsque les graphes OEM représentent des relations, le langage Lorel est équivalent au langag relationnel du premier ordre FO.

4.2 Le langage UnCAL

Le langage UnCAL (pour Unstructured CALculus) [BDS95] est un langage de type lambda calcul défini conjointement au modèle de données Arbre UP. Ce langage offre la particularité d'intégrer la **récursion structurelle** [BTBN91], par exemple pour parcourir un arbre, le parcours étant ici guidé par la structure.

Le langage UnCAL utilise trois formes de fonctions récursives ext, text et text’ pour parcourir un arbre. La première fonction ne considère que le premier niveau d'un arbre, la deuxième parcourt un arbre dans son intégralité sans aucune condition darrêt et la dernière effectue un parcours conditionnel (arrêt possible).

1. La fonction récursive **ext** applique une fonction f : label X arbre—— arbre sur un arbre de la façon suivante :

ext ({}) = { }
ext({l:t}) = f(l t)
ext(t_1 ? t_2) = ext(t_1) ? ext(t_2)

Par exemple, si f(l, t) est de la forme if l = **motionpicture then t else** { } alors ext appliquée sur l'arbre de la figure 2 retourne les trois arbres dont la racine est la destination d'un arc portant l'étiquette **motionpicture.**

2. La fonction récursive **text** applique une fonction f : label X arbre—— arbre sur un arbre de la façon suivante :

text({}) = { }
text({l :t})= f(l, text(t))
text(t_1 ? t_2)= text(t_1) ? text(t_2)

Par exemple, pour f(l text(t)) égale à l . text(t) , la fonction text posée sur l'arbre de la figure 2 rend l'arbre plat suivant :
{ motionpicture movie title Destiny author . . . }

La composition de ext et text donnée par

text({ }) = { }
text({ l : t}) = { l } ? ext(text(t))
text(t_1 ? t_2) = text(t_1) ? text(t_2)

ext({ }) = { }
ext({l_1 : t) = {l :{l_1 : t } }
ext(t_1 ? t_2) = ext(t_1) ? ext(t_2)

et appliquée sur l'arbre de la figure 2 retourne l'ensemble des chemins partant de la racine.
{ motionpicture

motionpicture : movie
motionpicture : { movie : title } ,
motionpicture : { movie : { title : Destiny } } ,
.
motionpicture : { tvserie : { episode : 100 } } }

Cet exemple est intéressant parce qu'il montre que certaines requêtes qui nécessitent l'utilisation de variables de chemin dans Lorel peuvent être exprimées par UnCAL sans avoir recours à ce type de variables.

3. La fonction récursive text' applique une fonction f : label X arbre X arbre ——arbre sur un arbre de la façon suivante :
text'({})= { }
text'({l_1 : t})= f(l, t, text'(t))
text'(t_1 ? t_2)= text'(t_1) ? text'(t_2)

Si f(l t text'(t)) est de la forme { l : t } . text'(t) alors la fonction text' appliquée sur l'arbre de la figure 2, retourne tous les sous-arbres.

La récursion structurelle présente deux avantages majeurs : (i) elle a une sémantique claire sur les arbres avec cycles et (ii) elle est calculable en temps polynômial. **UnCAL** est muni d'une sémantique déclarative et d'une sémantique opérationnelle pour traiter les cycles. La sémantique opérationnelle consiste à mémoriser les appels récursifs pour éviter les boucles in½nies. La récursion structurelle offre également la possibilité de développer des techniques d'optimisation assez fines [BDHS96]. Lorsque les Arbres UP représentent des relations, le langage UnCAL est équivalent au langage FO. La fermeture transitive d'une relation représentée par un arbre n'est pas exprimable par UnCAL.

4.3 Le langage StruQL

Le langage StruQL [FFLS97] est un langage d'interrogation et de restructuration de données semistructurées modélisées par des graphes à la OEM bien qu'il ait été initialement développé pour l'interrogation et la restructuration des sites Web. La requête StruQL ci-dessous posée sur le graphe de la figure 1, retourne les films dirigés par Chahine.

```
where  p ___motionpicture movie___q
       q ___director ___ "Chahine"
       q ___title_____ r
collect  ChahineMovies( r)
```

L'évaluation de cette requête correspond à : (1) trouver tous les chemins réalisant l'expression de chemin **motionpicture.movie** ensuite valuer la variable de graphe **p** (resp. **q**) par les graphes enracinés aux origines (resp. destinations) de ces chemins. Ainsi dans l'exemple, à p est associé le graphe enraciné en n_1 et à q sont associés les graphes enracinés en n_5 et n_6. (2) Tester si le graphe q a un arc étiqueté par **director** dont la destination porte l'étiquette Chahine (q——director——"Chahine") et vérifier si q a un arc étiqueté par title (q—title —r). La variable de graphe r sera valuée par les sous-graphes de q accessibles à partir de la racine de q et ce par un arc étiqueté par title. Dans l'exemple on garde n_5. Ainsi à r est associé le graphe enraciné en n_8. (3) Construire une collection (ensemble de graphes) nommée **ChahineMovies** contenant les graphes r (collect **ChahineMovies**(**r**)).

Le langage StruQL est plus puissant que Lorel et UnCAL. Lorsque les graphes représentent des relations, le langage StruQL est équivalent au langage relationnel du premier ordre plus la fermeture transitive FO+TC. La fermeture transitive de la relation binaire R de la figure 7, est exprimée par StruQL comme suit.

```
input ( where r ——"R"——s, s——"A"—— x s——"B"——y
        create New( x) , New( y)
        link New( x) . "bogus1" . New( y)
        collect Chain( New( x)) )

        where Chain( n) _____* _____m
        collect TC( n, m)
```

Cette requête est composée de deux sous-requêtes. La première est placée dans la clause input. Elle construit le graphe de la figure 8 où la collection Chain est représentée par des traits pointillés. La fonction New(p) est une fonction de Skolem qui retourne le même nœud pour la même valeur de l'étiquette du nœud p. La clause link New(x)___"bogus1"___New(y) crée un arc étiqueté par "bogus1" entre New(x) et New(y) . La deuxième sous-requête utilise l'expression de chemin " * " pour trouver tous les nœuds m accesibles à partir des nœuds n de la collection Chain (Chain(n)___* ___m). Ensuite, elle construit une collection TC contenant les couples de

Figure 8: Graphe avec la collection Chain.

Figure 9: Les deux arbres simples de l'h-arbre de la figure 5.

nœuds n et m (collect TC(n m)).

4.4 Le langage WebOQL

Le langage WebOQL [AM98] est un langage d'interrogation et de restructuration du Web modélisé par des h-arbres. WebOQL est un langage fonctionnel mais sa syntaxe ressemble à celle de SQL. La requête WebOQL ci-dessous posée sur le graphe de la figure 5, retourne les films dirigés par Chahine.

```
Select unique [y.title]
from   x ? mPicture y ? x'
where  x motionpicture =" movie"
       y. director =" Chahine"
```

L'évaluation de cette requête correspond aux étapes suivantes : (1) valuer la variable x par les deux arbres simples de la figure 9; (2) étant donnée une valuation pour x, y est valuée par les arbres simples de **x**'. L'application de l'opérateur Prime sur la variable **x** (c.-à-d., x.') revient à supprimer l'unique arc qui sorte de la racine de **x.** L'expression **x. motionpicture** extrait le contenu du champ motionpicture de l'unique arc sortant de la racine de **x.**

Figure 10: La fermeture transitive d'une relation R représenté par un h-arbre.

Lorsque les h-arbres représentent des relations, le langage WebOQL est équivalent au langage FO+TC. La fermeture transitive de la relation binaire R représentée par l'h-arbre de la figure 10.a est exprimée comme suit.

```
select unique    [x A] as “Chain” ,
                 [url : x B] as x A
from x . R
|
select unique [x A, B : y url] as “TC” ,
from x ? “Chain” ,
y ?  x A via > ( )..
```

Cette requête est constituée de deux sous-requêtes. Leur composition est spécifiée par le symbole“|” . La première crée une page avec l'url “**Chain**” (voir la figure 10.b) contenant toutes les valeurs distinctes dans la colonne “A” et, pour chacune de ces valeurs, une page qui rassemble toutes les valeurs distinctes dans la colonne “B” . En d'autres termes, la première sous-requête établit le graphe de la relation binaire “R” . La deuxième sous-requête prend chaque valeur enregistrée dans la page “Chain” et en commençant à la page associée à cette valeur, elle traverse tous les chemins spécifiés par l'expression > (>)* (“ > ” abstrait n'importe quel arc externe et * est la fermeture de Kleene) contenant au moins un arc. Ainsi, la deuxième sous-requête calcule la fermeture transitive de la relation binaire R donnée par l'h-arbre TC.

4.5 Le langage Graph-Fixpoint

Dans la section 3.1.3 nous avons introduit le modèle non imbriqué db-graphe. Le langage Graph-Fixpoint interroge les données semistructurées modélisées par des db-graphes. Le formalisme employé pour définir Graph-Fixpoint utilise la logique du premier ordre FO plus des expressions de chemin. Ce formalisme a permis d'introduire à l'aide d'outils mathématiques (point fixe) la récursion.

Exemple 1. La requête ci-dessous, posée sur le graphe (juste le graphe non imbriqué enraciné au nœud n3) de la figure 6, retourne les chemins dont leurs nœuds terminaux sont les titres de films. La formule **X** ? **movie title** formalise les chemins X épellés par l'expression de chemin **movie title**

{ (X) | X ? movie title }

La requête suivante teste l'existence du film Destiny dans le graphe en question. La variable de données a abstrait les données de nœuds dans un db-graphe

{ | (? X)(?a) X ? movie title |? a ? a =“ Destiny” }

Définition 4.1 (Fixpoint). Soit ?(X_1 , . . . , s X_n , X_1 . . . , X_n) une formule de chemin avec 2 X n variables de chemin libres, s X_1 , . . . , s X_n sont des variables distinguées et X_1, . . . , X_n . Soit G un db-graphe. L'expression de point fixe $\mu^+_{s X_1 , \ldots , s X_n}$ (?) dénote la relation limite de la suite inflationniste (cumulative) { S_k } k ? 0 définie par :
1. S_0 = { (e,. . . , e) }[2]
2. S k +1 = S_k ? (S_k) où

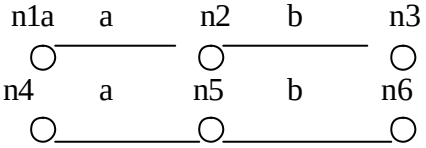
? (S k) est l'union de l'évaluation de la requête { (X_1 . . . , X_n) | ? [?$_k$] } sur G et ?$_k$: (X_1 , . . . , s X_n) —— (v1 . . . , v_n) ? S k .

Exemple 2. La requête qui retourne les paires de chemins de même séquence d'étiquettes d'arcs est

{ (X, Y) | $\mu^+_{sX\ sY}$ ((? x)(X? s_X x ? Y ? s_Y x))(X, Y) }

où X Y sont des variables de chemin et x y sont des variables d'étiquette d'arc. La formule X ? s_X x formalise les chemins X épellés par l'expression de chemin s_X x

Soit G le db-graphe suivant :



L'expression de point fixe $\mu^+_{s X , s Y}$ (?) dénote la relation limite de la suite inflationniste { S $_k$} k? 0 définie par :

S_0 = { (? ,?) }
S_1 = S_0 { ((n $_1$ n_2 a), (n$_4$ n_5 a)) ((n$_2$ n_3 b), (n$_5$ n_6 b)) }
S 2 = S 1 { ((n_1 n_2 a)(n_2 n_3 b), (n_4 n_5 a)(n_5 n_6 b)) }
S 3 = S 2

Exemple 3. Les chemins de longueur paire sont donnés par la requête

[2] (e,. . . , e) est un tuple de chemin vide d'arité n .

$$\{ (X) \mid \mu+_{s\,X} ((?\ x)(\ ?\ y)\ X\ ?\ s_X\ .\ x.\ y)(\ X\ ) \}$$

Soit G le chemin ($e_1\ e_2\ e_3\ e_4\ e_5$). L'expression de point fixe $\mu+_{s\,X}(?)$dénote la relation limite de la suite inflationniste $\{ S_k \}\ k\ ?\ 0$ définie par :

$S\ 0 = \{ (e) \}$

$S\ 1 = S_0\ .\ \{ e_1\ e_2\ ,\ e_2\ e_3\ ,\ e_3\ e_4\ ,\ e_4\ e_5 \}$

$S\ 2 = S_1\ .\ \{ e_1\ e_2\ e_3\ e_4\ ,\ e_2\ e_3\ e_4\ e_5 \}$

$S\ 3 = S_2$

Graph-Fixpoint permet de définir des prédicats intentionnels d'arité n quelconque sur des chemins, par exemple, le prédicat binaire SL qui teste l'égalité par valeur entre deux chemins peut être facilement défini comme suit :

$$\{ (X\ Y) \mid \mu+_{sX\ sY} ((?\ x)(\ ?\ y)(\ X\ \ ?\ sX\ x\ .\ Y\ \ ?\ \ s_Y\ y))(\ X,\ Y)\ \}$$

Il n'y a aucun autre langage manipulant des données semistructurées qui offre cette capacité; seuls des prédicats monadiques peuvent être définis sur l'ensemble des chemins.

Exemple 4 (Fermeture Transitive). La requête $\{ (X\ Y) \mid \mu+_{sX\ sY} (TC)(\ X\ Y)\ \}$ où la formule TC définie ci-dessous, retourne (sans compter le tuple de chemins vides) des paires de chemins où les paires de destinations respectives sont la fermeture transitive de la relation binaire R représentée par le db-graph de la figure 7. L'expression de la fermeture transitive n'a pas besoin de la création de nouveaux arcs comme c'était le cas dans StruQL et WebOQL. La formule TC est définie ci-dessous. La variable de données a dénote les données associées aux nœuds du db-graph de la figure 7 et l'expression SX a capture tout chemin dont la destination porte une donnée

$$(?\,U1)(\ U1?\ \text{“R”}\ \ ?\ \ X\ ?\ U_1\ \text{“A”}\ \ ?\ \ Y\ ?\ U_1\ \text{“B”})$$
$$?\ \ (?\ Z)(\ ?\ U_2)(\ ?a\ )(\ U2\ ?\ \text{“R”}\ \ ?\ \ \ X\ ?\ U_2\ \text{“A”}\ \ ?\ \ \ Z\ ?\ U_2\ \text{“B”}$$
$$?\ \ Z?\ a\ ?\ S_X\ a\ ?\ ?\ \ \ Y = S_Y)$$

5 Données semistructurées versus le modèle relationnel

Dans cette approche les graphes modélisant les données semistructurées sont codés par des relations. Des langages de type relationnel sont utilisés pour l'interrogation. Le langage WebCalculus [MM97] propose de modéliser le Web comme une base de données relationnelle avec deux relations : N et L. La relation $N(\ n,\ .\ .\ .\ ,\ a_i,\ .\ .\ .\ )$ a un tuple pour chaque document Web identifié par n et la relation $L(\ n\ m\ l_i....\ )$ a un tuple pour chaque lien hypertexte l_i d'un document n vers un document m. Cette modélisation a permis d'utiliser un langage de requête similaire à SQL ou au calcul relationnel. Une relation Path(n, R, m) est utilisée pour tester l'existence d'un chemin “simple” entre les deux documents n et m réalisant une expression régulière R.

Les auteurs de WebCalculus observent que certaines requêtes évaluables en temps polynômial telle que « Donner tous les document qui n'ont pas d'arc entrant » ne sont pas Web calculables, i.e., ne sont pas calculables dans le contexte du Web, parce que la seule façon d'accéder au Web est la navigation en suivant des liens à partir de(s) point(s) de départ. On ne peut pas savoir si un document n'a pas d'arc entrant parce qu'on ne peux pas avoir tous les points de départ sur le Web. Afin de limiter WebCalculus aux seules requêtes Web évaluables en temps polynômial, deux restrictions syntaxiques sont introduites. Premièrement, les premiers arguments de N, L et **Path** doivent être bornés. Cette restriction est appelée source safety. Deuxièmement et comme pour le calcul relationnel sain, il faut s'assurer que les

arguments d'un atome négatif sont bornés et que les termes d'une disjonction utilisent le même ensemble de variables libres. Le langage

Langage	Modèle de données	Style Puissance	d'expression
Lorel	Graphe	SQL	FO
UnCAL	graphe	calcul fonctionnel	FO
StruQL	graphe	SQL	FO+TC
WebOQL	h-arbre	SQL	FO+TC
WebCaluclus	relationnel	calcul logique	FO
Graph-Fixpoint	db-graphe	calcul logique	Fixpoint

Figure 11: Comparaison des langages de requêtes.

WebCalculus est équivalent au langage FO lorsque les graphes modélisés par N et L représentent des bases de données relationnelles.
Le langage WebLog [LSS96] est un langage d'interrogation du Web inspiré du Datalog et qui utilise des règles et des règles récursives pour exprimer des requêtes qui sont exprimées par ailleurs par des expressions régulières. Dans [AV97], un langage appelé ss-Datalog (source safe Datalog) permettant d'exprimer des requêtes Web évaluables en temps polynômial, a été proposé. Les auteurs de [AV97] prouvent que ss-Datalog avec une sémantique inflationniste permet d'exprimer toutes les requêtes de navigation (chaînage en avant). Florid [LHL + 98] est un dialecte du langage F-logic [KL89] qui permet d'interroger le Web en utilisant des règles. WebLog, ss-Datalog et Florid ont la même puissance d'expression que WebCalculus.
Les langages proposés dans la littérature expriment au mieux des requêtes qui sont dans la classe de requêtes FO+TC lorsque les graphes représentent des relations. Le langage de requêtes Graph-Fixpoint [BY98, BY99, Ykh99] que nous avons proposé exprime des requêtes dans la classe Fixpoint [AHV95] donc qui est strictement plus puissant que les langages existants. Le tableau de la figure 11 donne une comparaison générale de différents langages d'interrogation présentés dans cet article. La puissance d'expression est donnée en terme de classes de requêtes relationnelles posées sur des bases de données relationnelles modélisées par des graphes (voir figure 7).

6 Les expressions de chemin au cœur de l'histoire

La plupart des langages d'interrogation de données semistructurées proposés utilisent des expressions de chemin pour permettre la description de parcours dans les graphes sans pour autant avoir une connaissance précise de leur structure. Les expressions de chemin ont été utilisées premièrement dans les bases de données orientées objet [Cat97]. La manière la plus simple d'utilisation des expressions de chemin est de concaténer les étiquettes d'arc comme dans l'expression **motionpicture movie director**. Cette expression permet de naviguer dans le graphe de la figure 1 sans avoir à savoir si un film à un ou plusieurs directeurs.

Trois extensions significatives ont été proposées pour augmenter les possibilités d'utilisation des expressions de chemin.

1. Variables de chemin : les expressions de chemin utilisées dans les langages d'interrogation de données semistructurées utilisent des variables de chemin pour assouplir le mécanisme d'interrogation; une connaissance totale du chemin à parcourir n'est plus requise.

L'expression de chemin **X. title** où la variable **X** est une variable de chemin, spécifie tout chemin (peut être infini en présence de cycles) dans le graphe de la figure 1 amenant à un arc étiqueté par title. D'autre types de variables sont parfois introduits pour abstraire les étiquettes des arcs et les données associées aux nœuds.

2. Fermeture de Kleene : la fermeture de Kleene (# . #) * de l'expression de chemin # # où # est un joker remplaçant n'importe quelle étiquette permet de décrire tout chemin de longueur paire. L'expression de chemin (a b c) * abstrait des chemins de la forme abc, **abcabc, abcabcabc** . . .

3. Point fixe : dans Graph-Fixpoint [BY98, BY99] nous proposons d'étendre les expressions de chemin par un opérateur de point fixe. Cette extension a permis de :

(a) simuler la fermeture de Kleene,
(b) pouvoir définir des prédicats intentionnels d'arité n , et
(c) exprimer toutes les requêtes de point fixe dont la fermeture transitive. La fermeture transitive est exprimée sans avoir recours à de nouveaux arcs ajoutés au graphe d'origine comme c'est le cas dans les deux langages StruQL et WebOQL.

7 Conclusion

Dans cet article, nous avons étudié les langages d'interrogation de données semistructurées . Le modèle XML apparaît maintenant comme un standard incontournable pour l'échange des données sur le Web. De nombreux travaux cherchent à concevoir un langage d'interrogation et des techniques d'optimisation appropriées. Les recherches [Ykh99] sur les données semistructurées devraient se révéler utiles pour augmenter la puissance d'expression des langages d'interrogation pour XML et ajouter la possibilité d'imbriquer les données XML ainsi que les requêtes.

Bibliographies

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.

[AM98] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases, and webs.
In Proceedings of IEEE International Conference on Data Engineering (ICDE), Orlando, February 1998.

[AQM + 97] 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.

[AV97] S. Abiteboul and V. Vianu. Queries and computation on the web. In Proceedings of International Conference on Database Theory (ICDT), Delphi, Greece, January 1997.

[BDHS96] P. Buneman, S. B. Davidson, G. Hillebrand, and D. Suciu. A query language and opti-mization techniques for unstructured data. In Proceedings of ACM SIGMOD Conference on Management of Data, pages 505_516, Montreal, Canada, June 1996.

[BDS95] P. Buneman, S. B. Davidson, and D. Suciu. Programming constructs for unstructured data.
In Proceedings of International Workshop on Database Programming Languages, Gubbio, Italy, 1995.

[BMY99] N. Bidoit, S. Maabout, and M. Ykhlef. Un Langage Imbriqué pour l'interrogation et l'intégration des données semistructurées. In 15èmes Journées Bases de Données Avancées, Bordeaux, France, 25_28 October 1999.

[BMY00] N. Bidoit, S. Maabout, and M. Ykhlef. A Family of Nested Query Languages for Semi-structured Data. Lecture Notes in Computer Science, 1762:13_30, 2000.

[BPSM98] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml) 1.0.W3C Recommendation, February 1998.
http://www.w3.org/TR/REC-xml/.

[BTBN91] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In Proceedings of International Workshop on Database Programming Languages, pages 9_19, Nafplion, Greece, 1991. Morgan Kaufmann.

[BY98] 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.

[BY99] Nicole Bidoit and Mourad Ykhlef. Fixpoint Calculus for Querying Semistructured Data.Lecture Notes in Computer Science, 1590:78_98, 1999.

[Cat97] R. G. Cattell. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, 1997.

[CGMH + 94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeÿrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In 16th Meeting of the Information Processing Society of Japan, pages 7_18, Tokyo, Japan, 1994.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13(6):377_387, June 1970.

[FFLS97] M. Fernandez, D. Florescu, A. Y. Levy, and S. Suciu. A query-language and processor for a web site management system. In Workshop on Management of Semistructured Data, Tucson, Arizona, May 1997. In conjunction with PODS/SIGMOD.

[KL89] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In Proceedings of ACM SIGMOD Conference on Management of Data, volume 18, pages 134_146, Portland, Oregon, June 1989.

[LHL + 98] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. Information Systems, 23(8):589_612, 1998.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the WEB. In Proceedings of Sixth International Workshop on Research Issues in Data Engineering, pages 12_21, New Orleans, Louisiana, February 1996.

[Mil89] R. Milner. Communication and concurrency. Prentice Hall, 1989.

[MM97] A. Mendelzon and T. Milo. Formal models of web queries. In Proceedings of ACM Symposium on Principles of Database Systems, pages 134_143, Tucson, Arizona, May 1997.

[PGW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In Proceedings of IEEE International Conference on Data Engineering (ICDE), pages 251_260, Taipei, Taiwan, March 1995.

[QRS + 95] D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, and J. Widom. Querying semistructured heterogeneous information. In Proceedings International Conference on Deductive and Object-Oriented Databases (DOOD), volume 1013 of Lecture Notes in Computer Science, pages 319_344. Springer-Verlag, Singapore, December 1995.

[Ykh99] M. Ykhlef. Interrogation des données semistructurées. Thèse de doctorat, Université Bordeaux I, November 1999.