(Synthèse)
Contraintes d'intégrité pour XML : visite guidée par une syntaxe homogène

Béatrice Bouchou[*] **Mírian Halfeld Ferrari**[*] **Maria Adriana Lima**[*,**]

[*]*Université François Rabelais de Tours - Campus Blois - Laboratoire d'Informatique*
[**]*Pontifícia Universidade Católica de Minas Gerais - Poços de Caldas - Brésil*

{beatrice.bouchou, mirian.halfeld}@univ-tours.fr, adriana@pucpcaldas.br

RÉSUMÉ. *Le format XML sert à représenter des informations. Ces informations sont soumises à certaines contraintes d'intégrité, qu'il est utile de spécifier pour garantir la qualité des informations (éviter les incohérences). Les contraintes d'intégrité ont été beaucoup étudiées dans le cadre des bases de données relationnelles et, depuis le début des années 2000, des équivalents dans le cadre XML sont explorés par plusieurs équipes, qui ont focalisé leur attention sur un type ou un autre de contrainte. Dans cet article nous offrons un panorama sur les contraintes d'intégrité les plus courantes, à savoir les dépendances, en proposant un formalisme homogène et en explicitant leur sémantique.*

ABSTRACT. *The XML format is used to represent real-world data, which must respect certain integrity constraints. These constraints are defined to ensure accuracy and consistency of data. Within the framework of relational databases, integrity constraints have been extensively studied and since the beginning of 2000's, their XML equivalents are explored by several researches, which focused their attention on a type or another of constraint. This article aims to be a survey on XML integrity constraints by providing a homogeneous formalism and by clarifying the semantics of each type of constraint.*

MOTS-CLÉS : *XML, contraintes d'intégrité (clé, clé étrangère, dépendance fonctionnelle, dépendance d'inclusion)*

KEYWORDS: *XML, integrity constraints (primary key, foreign key, functional dependency, inclusion dependency)*

2e soumission à *TSI*, le 31 mars 2008

1. Introduction

Les contraintes appelées *contraintes d'intégrité* ont été introduites dans le cadre des bases de données relationnelles (et objets) pour exprimer le plus fidèlement possible les contraintes des systèmes d'information. Ces contraintes représentent donc des propriétés qui doivent être satisfaites par toutes les instances d'une base de données. Les premiers travaux de recherche dans ce domaine ont proposé l'expression des contraintes par des formules de la logique du premier ordre, mais pour que les systèmes de gestion de bases de données puissent les vérifier automatiquement de façon efficace, ces recherches se sont orientées vers l'étude de classes plus restreintes, couramment appelées *dépendances*. Par exemple, dire que le numéro de sécurité sociale (*NSS*) est la clé d'une relation EMPLOYEE est une contrainte d'intégrité qui impose l'existence et l'unicité des valeurs *NSS* dans toute la relation EMPLOYEE. Les contraintes de clés sont un cas particulier de contraintes plus générales appelées dépendances fonctionnelles. Un exemple d'une dépendance fonctionnelle qui n'est pas forcément une clé est la contrainte imposant qu'une même adresse ne puisse avoir qu'un seul code postal (exprimée par l'expression $Address \rightarrow ZIPCode$). Les dépendances d'inclusion représentent un autre type de contrainte d'intégrité très important. Par exemple, en supposant que la relation EMPLOYEE comprend le département *Dept* de chaque employé et qu'une table DEPARTEMENT stocke des informations pour chaque département de l'entreprise, une contrainte d'inclusion peut imposer que toute valeur *Dept* stockée dans EMPLOYEE doit exister dans DEPARTEMENT. De cette manière le système d'information reste cohérent et en ceci les contraintes d'intégrité sont un aspect essentiel de la *qualité des données*.

Nous savons également que les contraintes d'intégrité jouent un rôle tout aussi essentiel au niveau de la qualité des *schémas* de données, en servant de base à la théorie des formes normales. La possibilité de raisonnement formel sur les contraintes d'intégrité (Beeri *et al.*, 1979, Maier, 1980, Abiteboul *et al.*, 1995, Levene *et al.*, 1999) est un facteur de qualité pour les bases de données.

Un autre avantage qu'il y a à pouvoir définir des contraintes d'intégrité est qu'elles représentent des informations qui mènent à des implantations de stockage, de requêtes et de mises à jour plus efficaces. Par exemple, on formulera toute requête portant sur une classe (ou une table) en utilisant son identifiant, afin d'optimiser l'exécution de la requête. De même, on utilisera l'information fournie par une clé étrangère pour exprimer la jointure nécessaire à l'interrogation simultanée de deux relations.

XML (Extensible Markup Language (Bray *et al.*, 2004, Abiteboul *et al.*, 2000)) est devenu le standard de représentation des données du Web. Dans le cadre XML, les principes de conception de documents XML (ou de schémas de documents XML), de cohérence des données XML et de maintien de cette cohérence sont encore immatures. Ce n'est pourtant pas les besoins qui manquent, dans

la mesure où ce format s'est désormais imposé en particulier pour l'échange de données et les services Web. Les contraintes d'intégrité expriment une réalité : cette réalité doit être représentée et exploitée dans le format XML.

L'étude des contraintes d'intégrité pour XML a commencé dans les années 2000. Les premiers résultats démontrent à la fois l'intérêt qu'il y a à exprimer des contraintes d'intégrité sur les données XML (spécification), et l'intérêt de les définir formellement (analyse et application). La plupart des contraintes d'intégrité ont été abordées dans le cadre XML, avec leur analyse (implication, axiomatisation) et leurs applications (Fan, 2005). Nous revenons sur ces travaux dans la section 5.

Concernant la spécification de contraintes d'intégrité pour XML, le formalisme XIC (Deutsch *et al.,* 2005) est à notre connaissance le plus riche ; une contrainte d'intégrité y est définie ainsi : $\forall x_1, \ldots, x_n\ A(x_1, \ldots, x_n) \rightarrow \exists y_1, \ldots, y_m\ B(x_1, \ldots, x_n, y_1, \ldots, y_m)$, où x_i et y_j dénotent des variables dont les valeurs sont les noms des éléments ou attributs du document, et A et B sont des conjonctions d'atomes $x = y$ ou $x\ path\ y$, *path* définissant une relation de chemins entre x et y. Comme l'indiquent les auteurs, le pouvoir d'expression entraîne des problèmes d'indécidabilité pour des questions comme l'inclusion (ce qui est déjà le cas dans le cadre relationnel).

A l'opposé, les formalismes tels les DTDs (Bray *et al.,* 2004), XML-Schema (Thompson *et al.,* 2006) ou Relax-NG (Clark *et al.,* 2001), conçus pour la définition de contraintes de structure, offrent peu de constructions pour la spécification de contraintes d'intégrité. Les limites des attributs $ID/IDREF(S)$ d'une DTD sont évidentes : un attribut *ID* doit avoir une valeur unique, mais parmi tous les attributs *ID* du document XML, ce qui empêche par exemple des composants et des fournisseurs d'avoir des identifiants identiques. De même, une référence *IDREF* peut correspondre à n'importe quel *ID* dans le document. De son côté, Relax-NG ne traite que de la structure, contrairement au langage XML Schema, qui permet effectivement de définir des clés et clés étrangères .

Dans cet article, nous réunissons dans un formalisme homogène les propositions actuelles concernant les principales contraintes d'intégrité pour XML, les dépendances, en définissant également l'interprétation de chaque type de contrainte, à la fois formellement et intuitivement.

L'article est organisé de la façon suivante : dans la section 2 nous présentons les notions nécessaires pour la définition des dépendances et la formalisation de leur satisfaction. La section 3 est consacrée aux dépendances fonctionnelles et dépendances d'inclusion (plus les dépendances de lien inverse), y compris dans le cas d'informations incomplètes. La section 4 présente les clés et les clés étrangères, ainsi que les principes de leur vérification automatique. La section 5 permet de présenter les travaux de ce domaine, avant de conclure.

2. Préliminaires

Dans cet article, nous considérons un document XML comme une structure T composée d'un arbre t d'arité non bornée (représentant la structure du document) et de deux fonctions *type* et *value*. La fonction *type* indique si un nœud de l'arbre représente un *élément*, un *attribut* ou une donnée *data* et la fonction *value* retourne la valeur liée à une feuille (un nœud de type *attribut* ou *data*). Par exemple, la figure 1 représente un document XML sous la forme d'un arbre. Les noms des attributs sont précédés du symbole @ et les valeurs sont en italique.

L'arbre t est une fonction d'un ensemble fini de positions (le *domaine*) vers un ensemble d'étiquettes Σ. Une position est un identifiant unique d'un nœud de t et est représentée par une séquence d'entiers positifs. Par exemple, dans la figure 1 le premier élément de label *fournisseur* est en position 0.1. On constate dans cette figure que chaque position est toujours préfixée par celle de son père et correspond à la séquence de nœuds qui permet d'accéder à ce nœud précis, depuis la racine. Dans la suite nous confondrons les notions de nœud et de position.

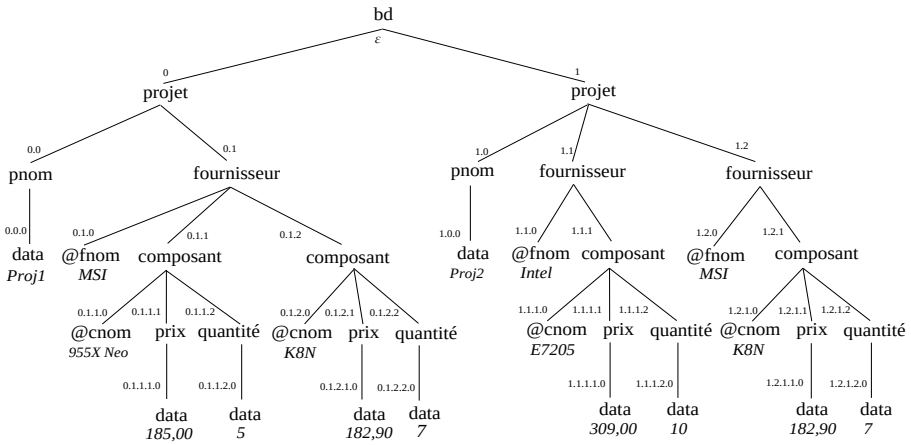

Figure 1. *Document XML T décrivant une liste de projets.*

Définition 2.1 - Domaine d'un arbre - Notons U l'ensemble des entiers positifs, U^* est l'ensemble des mots de symboles dans U, la concaténation étant dénotée par le point. La séquence vide, notée ε, est associée à la racine de l'arbre. Le *domaine* de l'arbre t, noté $dom(t)$, est l'ensemble des positions de t, *fermé par préfixe*[1] et qui satisfait : $(\forall j \geq 0)\ (u.j \in dom(t)) \Rightarrow (\forall i\ \ 0 \leq i \leq j)\ (u.i \in dom(t))$ où i et $j \in U$. □

1. Si $v \in dom(t)$ alors $\forall u$ tel que $u.w = v$, $u \in dom(t)$ (tout préfixe de v est dans $dom(t)$).

Intuitivement, la dernière condition indique que si la position 1.2 existe, alors les positions 1.0 et 1.1 (les frères gauches) existent aussi.

Définition 2.2 - Représentation arborescente d'un document XML - Soit un alphabet $\Sigma = \Sigma_{ele} \cup \Sigma_{att} \cup \{data\}$, dans lequel Σ_{ele} est l'ensemble des noms d'éléments et Σ_{att} est l'ensemble des noms d'attributs. Un document XML est représenté par un tuple $T = (t, type, value)$ tel que :

– l'arbre t est une fonction : $t : dom(t) \rightarrow \Sigma \cup \{\lambda\}$

Pour chaque $p \in dom(t)$, $t(p) = a$ indique que le symbole $a \in \Sigma$ est associé au nœud de position p. La racine est associée à la position ε et l'arbre vide est défini par $t = \{(\varepsilon, \lambda)\}$ où $\lambda \notin \Sigma$ est un symbole réservé.

– Les fonctions *type* et *value* sont définies pour toute position $p \in dom(t)$:

$$type(p) = \begin{cases} data & \text{si} \quad t(p) = data \\ \textit{élément} & \text{si} \quad t(p) \in \Sigma_{ele} \\ attribut & \text{si} \quad t(p) \in \Sigma_{att} \end{cases}$$

et $value(p) = p$ si $type(p) = \textit{élément}$; sinon $value(p) = val \in \mathbf{V}$, où $\mathbf{V}$ est un ensemble récursivement énumérable. □

Comme t est un arbre, nous pouvons utiliser les fonctions classiques sur des arbres : pour tout nœud de position p dans $dom(t)$, $fils(p)$ est l'ensemble des nœuds fils de la position p et $père(p)$ est le noeud père de p. Par exemple dans la figure 1, $fils(1) = \{1.0, 1.1, 1.2\}$, et $père(0) = \varepsilon$.

2.1. *Désigner des parties d'un document*

Plusieurs langages de chemin sont possibles pour désigner des parties d'un document XML, allant de la séquence d'étiquettes (Buneman *et al.,* 2001a) à des langages plus expressifs comme XPath (Clark *et al.,* 1999). Nous utilisons le langage de chemin *PL* défini ci-après.

Définition 2.3 - Langage de chemins *PL* **-** Un chemin en *PL* respecte la syntaxe suivante :

$$v ::= [\,] \mid q \mid v//q$$
$$q ::= l \mid q/q \mid _$$

où [ ] représente le chemin vide, l est une étiquette dans $\Sigma_{ele} \cup @\Sigma_{att} \cup data$, le symbole "/" est l'opération de concaténation, "//" représente n'importe quelle séquence finie (éventuellement vide) d'étiquettes de nœuds, et "_" représente une étiquette quelconque dans Σ. □

On appelle *chemins simples* des chemins qui sont de simples séquences d'étiquettes (sans //, ni _). Dans le cas de chemins simples, on peut utiliser les

notions suivantes. Soit un chemin $P = l_1/\ldots/l_n$, où $n \geq 1$. Le dernier pas de P est obtenu par la fonction *Last* : $Last(P) = l_n$. La fonction *Parent* retourne le chemin égal à P sans son dernier pas : $Parent(P) = l_1/\ldots/l_{n-1}$. Soit Q le chemin $q_1/\ldots/q_m$. Le chemin P est dit un préfixe du chemin Q, noté $P \subseteq Q$, si $n \leq m$ et $l_1 = q_1, \ldots, l_n = q_n$. Deux chemins P et Q sont égaux, ce qui se note $P = Q$, si P est un préfixe de Q et Q est un préfixe de P. Le chemin P est un préfixe strict du chemin Q, noté $P \subset Q$, si P est un préfixe de Q et $P \neq Q$.

Par exemple, pour le chemin $P_1 = bd/projet/fournisseur$, la fonction $Parent(P_1)$ retourne le chemin $bd/projet$, et si on considère le chemin $Q = bd/projet/fournisseur/composant/@cnom$, alors $P_1 \subseteq Q$.

Nous pouvons faire un parallèle entre schéma et chemin. Un schéma (DTD, XML-Schema, ou Relax-NG) établit la structure d'un document XML. Un arbre XML valide par rapport à un schéma est une instance de ce schéma. Un chemin P dans le langage *PL* peut être vu comme un schéma très réduit, qui définit un parcours vertical et sans branchement sur un arbre XML. Réciproquement, un arbre XML est une collection d'instances de chemins qui partent de la racine. Étant donné un chemin P et un document T, instancier P par rapport à T signifie trouver les séquences de nœuds père-fils (que nous appelons *segments*) dans t qui respectent P. Il peut y avoir zéro ou plusieurs instances d'un chemin P dans T. Par exemple dans la figure 1, il y a 3 instances du chemin $bd/projet/fournisseur$, qui sont $\varepsilon/0/0.1$, $\varepsilon/1/1.1$ et $\varepsilon/1/1.2$.

Définition 2.4 - Segment - Un segment dans un document T est un couple de positions (v_o, v_e) de $dom(t)$ tel que v_o est un préfixe de v_e. Un segment $s = (v_o, v_e)$ est un *préfixe* d'un segment $s' = (v'_o, v'_e)$ si $v_o = v'_o$ et v_e est un préfixe de v'_e. La fonction *LastPosition* appliquée à $s = (v_o, v_e)$ retourne v_e. On note $Segments(T)$ l'ensemble des segments de T qui partent de la racine ($v_o = \varepsilon$). □

Le langage *PL* nous permet de construire des chemins selon une syntaxe correspondant à des cas simples d'expressions régulières. Soit P un chemin dans *PL*, on définit A_P l'automate à états finis qui reconnaît l'ensemble des chemins simples représentés par P de la façon suivante : $A_P = \{Q, \Sigma, I, \delta, F\}$, où

– Q est un ensemble fini d'états,
– Σ = {étiquettes de nœuds},
– I contient un seul état initial,
– δ représente la séquence des étiquettes de P, où chaque // est représenté par une transition *.
– F contient un seul état final.

On peut alors associer à un segment $s = (v_o, v_e)$, qui représente un mot de positions $v_1 \ldots v_n$ où $v_1 = v_o$ et $v_n = v_e$, le chemin (simple) d'étiquettes correspondant de la façon suivante : $etiquettes(s) = t(v_1) \ldots t(v_n)$. Ainsi, trouver les instances d'un chemin P sur un document T revient à trouver les segments

s de T tels que $etiquettes(s) \in L(A_P)$[2]. Par exemple, on verra dans la figure 6 une représentation des automates des chemins $bd/projet$, $fournisseur$, et $@fnom$.

Définition 2.5 - Instance d'un chemin P dans un document T - Soit P un chemin, soit A_P l'automate à états finis correspondant à P et soit T un document. Un segment $s = (v_o, v_e)$ dans T est une instance de P sur T si et seulement si $etiquettes(s) \in L(A_P)$. On note $Instances(P, T)$ l'ensemble de toutes les instances d'un chemin P dans un document T. □

Les contraintes d'intégrité sont définies par un ensemble de chemins, chacun déterminant une partie du document concernée par la contrainte. Intuitivement, les instances de ces chemins doivent être positionnées dans l'arbre du document *de façon à former une arborescence qui parte de l'élément racine du document* (les parties concernées se déterminent relativement à la racine). Pour formaliser le lien entre contrainte et document (définition 2.7), nous introduisons la notion de *motif*, correspondant à un ensemble M de chemins.

Définition 2.6 - Motif - Un *motif* est formé d'un ensemble fini de chemins, *fermé par préfixe*. □

Exemple 2.1 *L'ensemble de chemins $M = \{bd,\ bd/projet,\ bd/projet/pnom,\ bd/projet/pnom/data,\ bd/projet/fournisseur,\ bd/projet/fournisseur/@fnom\}$ est un motif.* □

Cette notion de *motif* induit une structure, qu'on s'attend à trouver dans un document XML. L'avantage d'utiliser un ensemble de chemins, plutôt qu'un schéma comme une DTD est de se situer à un niveau plus simple, sachant que la conformité d'un document XML à un ensemble de chemins est moins restrictive que la conformité par rapport à une DTD (Vincent *et al.*, 2004a) (il n'y a pas de notion d'ordre entre les chemins, ni de cardinalité (répétition d'un chemin), ni d'option (disjonction dans les expressions régulières d'un schéma)). La conformité d'un document par rapport à un motif est une condition suffisante pour la vérification des contraintes d'intégrité : c'est la démarche adoptée dans (Buneman *et al.*, 2001a), qui permet de spécifier des contraintes d'intégrité indépendamment d'un schéma. Dans (Arenas *et al.*, 2002b), les auteurs se servent de l'ensemble de tous les chemins dans une DTD D, noté *Paths(D)*. C'est aussi grosso-modo ce que représente le *graphe-schéma* utilisé dans (Hartmann *et al.*, 2006) et l'ensemble de *chemins légaux* de (Liu *et al.*, 2003, Vincent *et al.*, 2004a). Nous revenons sur ces références dans la section 5. En résumé, pour formaliser la notion de contrainte d'intégrité, il faut pouvoir exprimer que T est conforme à une structure X (moins contraignante qu'un schéma XML), pour indiquer qu'il existe dans T des instances pouvant vérifier la contrainte. Cette condition est notée $T \triangleleft X$, X pouvant être, selon les approches, *Paths(D)*, un *ensemble de chemins légaux*, un *graphe schéma* ou un *motif*.

2. $L(A_P)$ dénote le langage reconnu par A_p.

Définition 2.7 - Conformité, motif complet - Soit M un *motif* et soit $T = (t, value, type)$ un document XML. T est *conforme au motif* M ($T \triangleleft M$) si pour tout $I \in Segments(T)$, il existe $P \in M$, tel que $I \in Instances(P, T)$. De plus, si pour tout chemin $P \in M$, il existe un segment $I \in Segments(T)$ tel que $I \in Instances(P, T)$, le motif M est un *motif complet* pour T. □

On remarque que le fait qu'un motif soit fermé par préfixe sert dans cette définition de la conformité.

Exemple 2.2 *M_1 est un motif complet pour le document T de la figure 1, et $T \triangleleft M_1$.*

$$\begin{aligned} M_1 = \{ \; & bd, bd/projet, bd/projet/pnom, bd/projet/pnom/data, \\ & bd/projet/fournisseur, bd/projet/fournisseur/@fnom, \\ & bd/projet/fournisseur/composant, \\ & bd/projet/fournisseur/composant/@cnom, \\ & bd/projet/fournisseur/composant/prix, \\ & bd/projet/fournisseur/composant/prix/data, \\ & bd/projet/fournisseur/composant/quantité, \\ & bd/projet/fournisseur/composant/quantité/data \; \} \end{aligned}$$ □

Définition 2.8 - Instance d'un motif M dans un document T - Soient un document $T = (t, type, value)$ et un motif M. On note $\mathcal{I} = (t^i, type^i, value^i)$ une *instance de M dans T*, où l'ensemble des positions de t^i, noté Δ, est inclus dans $dom(t)$, de plus t^i, $type^i$ et $value^i$ sont les restrictions de t, $type$ et $value$ à Δ, et enfin $\mathcal{I}$ vérifie toutes les conditions suivantes :

1) $\varepsilon \in \Delta$,
2) $\mathcal{I}$ est conforme à M,
3) Pour tout chemin $P \in M$, il existe une et une seule instance de P dans $\mathcal{I}$,
4) Δ est fermé par préfixe. □

Notons qu'une instance $\mathcal{I}$ d'un motif ne correspond pas exactement à la définition de document XML car Δ ne respecte pas la définition d'un domaine (cf. définition 2.1). Mais par abus de langage, nous parlons d'instance d'un chemin P dans une instance de motif $\mathcal{I}$ pour désigner l'instance p^i du chemin P dans T telle que p^i est un segment de $\mathcal{I}$.

Définition 2.9 - Coïncidence de deux instances de motif sur un chemin P - Soient deux instances du motif M dans un document T, $\mathcal{I} = (t^i, type^i, value^i)$ et $\mathcal{J} = (t^j, type^j, value^j)$, nous disons que $\mathcal{I}$ et $\mathcal{J}$ *coïncident sur un chemin P* si et seulement si il existe une instance p^i de P dans $\mathcal{I}$ et il existe une instance p^j de P dans $\mathcal{J}$ et $p^i = p^j$. □

Exemple 2.3 *Pour le motif M de l'exemple 2.1 et pour le document T de la figure 1, une instance de M envisageable est $\mathcal{I}$ telle que $t^i = \{(\varepsilon, bd), (1, projet), (1.0, pnom),$*

$(1.0.0, data)$, $(1.1, fournisseur)$, $(1.1.0, @fnom)\}$. *On vérifie en effet qu'il y a dans $\mathcal{I}$ la position ε (condition 1), que tous les segments de $\mathcal{I}$ qui partent de la racine sont des instances de chemins de M (condition 2), qu'il y a une et une seule instance de tous les chemins de M (condition 3) et que l'ensemble des positions de $\mathcal{I}$ est fermé par préfixe.*

Par contre, la partie de document contenant l'ensemble de positions $\{\varepsilon, 0, 0.0, 0.0.0, 1.1, 1.1.0\}$ n'est pas une instance de M, car cet ensemble de positions n'est pas fermé par préfixe.

Une autre instance de M, notée $\mathcal{J}$, a pour ensemble de positions $\{\varepsilon, 1, 1.0, 1.0.0, 1.2, 1.2.0\}$: on note que $\mathcal{J}$ et $\mathcal{I}$ coïncident sur le chemin $P = bd/projet$. Une troisième instance de M, notée $\mathcal{K}$, a pour ensemble de positions $\{\varepsilon, 0, 0.0, 0.0.0, 0.1, 0.1.0\}$; $\mathcal{K}$ ne coïncident pas avec $\mathcal{I}$ (ou $\mathcal{J}$) sur P. □

La notion d'instance de motif est à rapprocher de la notion de projection utilisée dans (Benzaken *et al.*, 2006) : l'instance de motif est une projection du document T, le motif étant le "projecteur" correspondant aux "type projectors" définis dans cet article.

Pour rester dans le même cadre que la plupart des propositions concernant les contraintes d'intégrité en XML, nous utilisons un langage de chemins linéaires pour les définir. L'utilisation de requêtes arborescentes serait une autre option, qui offre un autre pouvoir d'expression. Par exemple, PL ne nous permet pas d'exprimer de manière distincte les arborescences $r(a(b,c), a(b,d))$ et $(r(a(b,c,d))$. L'extension du langage de requêtes utilisé pour l'expression des contraintes est donc une perspective de travail intéressante, explorée dans (Halfeld Ferrari Alves, 2007) où des requêtes d'arbres simples (de type *tree pattern* (Benedikt *et al.*, 2003)) sont utilisées comme une sorte d'abréviation pour représenter une collection de chemins exprimés dans un langage comme *PL*. Notons que, même si notre formalisme de définition des contraintes s'appuie sur un langage de chemins, les notions de motif et d'instance de motif nous conduisent à considérer des *arborescences dans le document*, puisque les instances d'une contrainte correspondent à des n-uplets de valeurs (comme le précisera la définition 3.2).

2.2. *Informations incomplètes*

Un document XML T représente couramment des informations incomplètes C'est le cas pour le document T de la figure 2, dans lequel un composant (position 0.2.1) n'a pas de fils *prix* ni de fils *quantite*. Le chemin $P = bd/projet/fournisseur/composant/prix$ possède une instance incomplète sur T, à savoir $\varepsilon/0/0.2/0.2.1$.

Le traitement correct des informations incomplètes est fondamental, aussi bien pour le calcul des requêtes que pour la vérification des contraintes d'intégrité. Différentes propositions, établies dans le cadre du modèle relation-

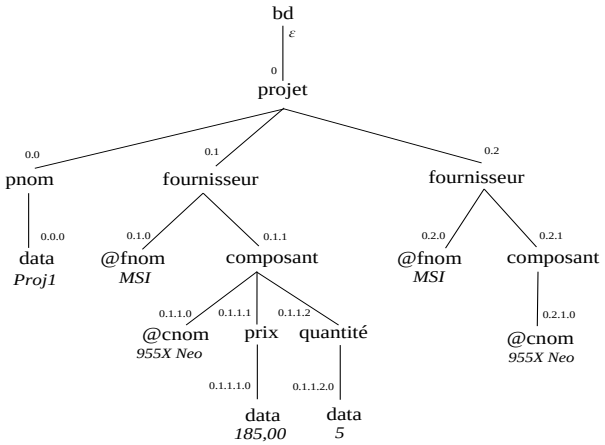

Figure 2. *Document XML avec des informations manquantes.*

nel (Codd, 1979, Lipski, 1981, Lien, 1982, Imieliński *et al.*, 1984, Atzeni *et al.*, 1986, Imielinski, 1989, Levene *et al.*, 1999), peuvent être transposées aux documents XML (Abiteboul *et al.*, 2001, Arenas *et al.*, 2002b, Vincent *et al.*, 2004a, Hartmann *et al.*, 2006). Dans la définition suivante, nous précisons la notion d'instance incomplète d'un chemin P sur un document T.

Définition 2.10 - Instance incomplète d'un chemin P sur un document T - Soit P un chemin dans le langage PL. Soit T un document XML. Un segment $s = (v_o, v_e)$ dans T est une *instance incomplète* de P sur T ssi il est une instance d'un préfixe strict de P (définition 2.5) qui respecte les conditions suivantes :
(i) $\hat{\delta}(e_0, t(v_o) \ldots t(v_e)) = e$ pour l'automate d'état fini A_P dont $\hat{\delta}$ est la fonction de transition étendue[3], e_0 est l'état initial et e est un état non final et
(ii) il n'existe aucun fils $v_e.i$ de v_e tel que $\delta(e, t(v_e.i))$ soit une transition de A_P. □

Nous étendons la notion d'instance de motif à celle *d'instance incomplète* de motif de façon analogue. Une instance de motif est incomplète si elle contient au moins une instance incomplète de chemin.

3. La fonction de transition étendue ((Hopcroft *et al.*, 2001)) décrit le résultat (l'état) obtenu via un parcours sur l'automate d'état fini qui commence par un état donné, en lisant une suite de symboles d'entrée. La fonction $\hat{\delta}$ est construite sur la fonction de transition δ : $\hat{\delta}(s, \epsilon) = s$ (si s est un état de l'automate et aucun symbole n'est lu) et $\hat{\delta}(s, w) = \delta(\hat{\delta}(s, x), a)$ (où $w = xa$).

2.3. *Tester si deux parties de document sont identiques*

Dans la section 2.1 (par exemple dans la définition 2.4) nous avons utilisé l'égalité pour indiquer que deux positions d'un arbre étaient les mêmes (égalité de nœuds). Dans cette section nous nous intéressons à une autre forme d'égalité : l'égalité par valeur.

Définition 2.11 - Égalité par valeur - Deux nœuds de positions p et q sont égaux en valeur, égalité notée $p =_V q$, si les conditions suivantes sont respectées :
(*i*) $t(p) = t(q)$ (ils ont la même étiquette) ;
(*ii*) $type(p) = type(q)$ (ils ont le même type) ;
(*iii*) si $type(p) = data$ ou $type(p) = attribut$, alors $value(p) = value(q)$ (ils ont la même valeur sur leur donnée) ;
(*iv*) si $type(p) = élément$, alors il existe une fonction bijective entre les fils de p et ceux de q qui associe chaque position $p.i$ à une position $q.j$ telle que $p.i =_V q.j$ □

Par exemple dans la figure 1, le nœud en position 0.1.2 est égal en valeur au nœud en position 1.2.1. La définition suivante permet d'indiquer que deux nœuds concordent sans préciser par quelle égalité.

Définition 2.12 - Concordance de nœuds - Soient deux nœuds n_1 et n_2 de même étiquette dans un arbre T. Les nœuds n_1 et n_2 concordent, ce qui se note $n_1 =_E n_2$, si et seulement si ou bien $n_1 =_V n_2$ (égalité en valeur, dans ce cas $E = V$), ou bien $n_1 = n_2$ (identité de nœuds, dans ce cas $E = N$). □

3. Dépendances fonctionnelles et dépendances d'inclusion

3.1. *Dépendances fonctionnelles*

Des dépendances fonctionnelles dans le contexte XML ont été définies dans (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a, Liu *et al.*, 2003, Hartmann *et al.*, 2006) et nous revenons sur ces propositions dans la section 5. Comme dans le modèle relationnel, une dépendance fonctionnelle pour un document XML (*XDF*) se note $X \rightarrow Y$ (où X et Y sont des ensembles de chemins) et impose que pour chaque paire de n-uplets[4] t_1 et t_2 si $t_1[X] = t_2[X]$ alors $t_1[Y] = t_2[Y]$. On peut structurer les *XDF* de façon à n'avoir qu'un seul chemin à droite (Arenas *et al.*, 2002b). La définition 3.1 généralise les propositions sur XML citées précédemment, dans le sens où (*i*) nous proposons la spécification d'un contexte pour la dépendance et (*ii*) nous proposons de définir plusieurs

Définition 3.1

4. Formés par les valeurs dans les extrémités des instances des chemins dans X et dans Y sur un document T.

chemins en partie gauche et un chemin en partie droite qui ne sont pas obtenus à partir d'une DTD ou d'un autre schéma XML.

- Dépendance fonctionnelle - Une dépendance fonctionnelle XML (*XDF*) est une expression de la forme

$$\gamma = (C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow Q\ [E]))$$

où C est un chemin qui représente *le contexte* dans lequel la dépendance fonctionnelle doit être respectée (la dépendance est alors dite *relative*), $\{P_1, \ldots, P_k\}$ est un ensemble non vide de chemins et Q est un chemin. Si $C = []$, alors la dépendance est dite *absolue* et on peut abréger l'expression par : $(\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow Q\ [E])$.

Les symboles $E_1, \ldots, E_k, E$ représentent le type d'égalité associé à chaque chemin dans la dépendance fonctionnelle : Si $E_i = V$ alors cette partie de la dépendance fonctionnelle doit être vérifiée avec l'égalité de valeurs. Si $E_i = N$ alors cette partie doit être vérifiée avec l'identité de nœuds. Quand les symboles E ou $E_1, \ldots, E_k$ sont omis, on utilise par défaut l'égalité de valeurs. De même, les chemins finissant dans un nœud *data* ou *attribut* ne peuvent être associés qu'à l'égalité de valeurs. □

Voici des exemples de *XDF*, qui expriment des dépendances pour des données comme celles contenues dans le document de la figure 1 :

XDF_1 : $(\{bd/projet/pnom\} \rightarrow bd/projet\ [N])$
Deux projets distincts (c'est-à-dire représentés par deux nœuds distincts) ne peuvent pas avoir le même nom. XDF_1 ne comportant pas de contexte, la dépendance doit être vérifiée dans tout l'arbre t.

XDF_2 : $(\{bd/projet/pnom\} \rightarrow bd/projet)$
Deux projets ayant le même nom doivent être égaux en valeur.

XDF_3 : $(\{bd/projet/fournisseur/@fnom,$
$bd/projet/fournisseur/composant/@cnom\}$
$\rightarrow\ bd/projet/fournisseur/composant/prix)$
Un fournisseur doit fournir un composant au même prix indépendamment des projets, dans tout le document.

XDF_4 : $(bd/projet, (\{/fournisseur/@fnom, /fournisseur/composant/@cnom\}$
$\rightarrow /fournisseur/composant/quantite))$
Dans chaque projet (contexte), un composant doit avoir une quantité unique.

XDF_5 : $(\{bd/projet\}\ [N]\ \rightarrow bd/projet/pnom)$
Un projet ne peut pas avoir deux numéros différents. Remarquer que cette contrainte n'impose pas à un nœud *projet* d'avoir un seul fil *pnom* (ce n'est pas une clé). Mais tous les *pnom* fils de *projet* doivent avoir la même valeur.

XDF_6 : $(bd/projet, (\{fournisseur\ [N], fournisseur/composant/@cnom\ [V]\})$
$\rightarrow fournisseur/composant/prix$

Dans un même projet, deux composants fournis par un même fournisseur et ayant le même nom, doivent avoir le même prix.

Une *XDF* définit des conditions que doivent vérifier les n-uplets formés des valeurs atteintes par les chemins spécifiés. La définition suivante précise les n-uplets considérés.

Définition 3.2 - N-uplet d'instance de motif - Soit M un motif et soit un ensemble de chemins $P = \{P_1, \ldots, P_k\}$ tel que $P \subset M$. Soit $\mathcal{I} = (t^i, type^i, value^i)$ une instance de M sur un document T. Selon la définition 2.8, $\mathcal{I}$ contient une et une seule instance de chaque P_j : on note I_j l'instance du chemin P_j dans $\mathcal{I}$, $\forall j \in [1 \ldots k]$. On note $\vec{P}$ le vecteur $P_1, \ldots, P_k$. Le n-uplet correspondant à $\vec{P}$ sur $\mathcal{I}$, noté $\mathcal{I}[\vec{P}]$, est tel que : $\mathcal{I}[\vec{P}] = [value^i(LastPosition(I_1)), \ldots, value^i(LastPosition(I_k))]$.
On note $\mathcal{I}[P_j]$ la valeur $value^i(LastPosition(I_j))$. □

Le n-uplet $\mathcal{I}[\vec{P}]$ est l'ensemble des valeurs obtenues d'un document T à partir d'une instance d'un motif M, et est construit selon la perspective nommée des bases de données relationnelles (Abiteboul *et al.*, 1995) (c'est-à-dire que les noms des attributs composant le n-uplet sont connus). Deux n-uplets $\mathcal{I}[\vec{P}]$ et $\mathcal{J}[\vec{P}]$ sont égaux relativement à $\vec{E} = E_1, \ldots, E_k$, ce qui se note $\mathcal{I}[\vec{P}] =_{\vec{E}} \mathcal{J}[\vec{P}]$, ssi $\forall j \in [1 \ldots k], \mathcal{I}[P_j] =_{E_j} \mathcal{J}[P_j]$.

La définition ci-dessous formalise la sémantique des *XDF* avec les deux notions d'égalité, comme dans l'approche proposée dans (Wang *et al.*, 2005).

Définition 3.3 - Satisfaction de *XDF* **-** Soient $T = (t, type, value)$ un document XML et $\gamma = (C, (\{P_1\,[E_1], \ldots, P_k\,[E_k]\} \rightarrow Q\,[E]))$ une *XDF* . Soit M le motif formé à partir de $\{C/P_1, \ldots, C/P_k, C/Q\}$. T satisfait γ (et cela s'écrit $T \models \gamma$) ssi $\forall \mathcal{I}, \mathcal{J}$, instances de M dans T qui coïncident au moins sur le chemin C, on a :
$\mathcal{I}[C/\vec{P}] =_{\vec{E}} \mathcal{J}[C/\vec{P}] \Rightarrow \mathcal{I}[C/Q] =_E \mathcal{J}[C/Q]$ avec $C/\vec{P} = C/P_1, \ldots, C/P_k$. □

Exemple 3.1 *Considérons le document T de la figure 1 et les dépendances* XDF_1 *et* XDF_2*. Pour le motif M formé à partir de l'ensemble de chemins dans la partie gauche* $C/\vec{P} = \{bd/projet/pnom\}$ *et du chemin* $C/Q = bd/projet$*, nous obtenons dans T les deux instances* $\mathcal{I}$ *et* $\mathcal{J}$*, telles que* $\mathcal{I}[C/\vec{P}] = [Proj1]$ *et* $\mathcal{J}[C/\vec{P}] = [Proj2]$*. L'égalité* $\mathcal{I}[C/\vec{P}] =_{\vec{E}} \mathcal{I}[C/\vec{P}]$ *n'est donc pas vérifiée. Dans ce cas le document T respecte aussi bien* XDF_1 *que* XDF_2*.*

On vérifie de la même manière que XDF_3 *est également respectée, vu que dans le cas où un fournisseur (MSI) fournit un même composant (K8N) pour deux projets différents, le composant a le même prix.* XDF_4 *est aussi respectée puisque sous chaque projet, chaque fournisseur fournit une quantité unique par composant.*

Considérons maintenant la XDF suivante :
XDF_7 *:* $(\{bd/projet/fournisseur/@fnom\} \rightarrow bd/projet/fournisseur\,[N])$

(deux fournisseurs, dans deux nœuds distincts, ne peuvent pas avoir le même nom). Pour le motif M formé à partir de l'ensemble de chemins dans la partie gauche $C/\vec{P} = \{bd/projet/fournisseur/@fnom\}$ *et du chemin* $C/Q = bd/projet/fournisseur$*, il existe dans T trois instances* $\mathcal{I} = \{\epsilon/0/0.1/0.1.0, \epsilon/0/0.1\}$, $\mathcal{J} = \{\epsilon/1/1.1/1.1.0, \epsilon/1/1.1\}$ *et* $\mathcal{J}' = \{\epsilon/1/1.2/1.2.0, \epsilon/1/1.2\}$.

La dépendance XDF_7 *n'est pas respectée dans T car* $\mathcal{I}[C/\vec{P}] =_V \mathcal{J}'[C/\vec{P}]$ *mais* $\mathcal{I}[C/Q] \neq_N \mathcal{J}'[C/Q]$ *(il existe deux fournisseurs avec le même nom (MSI) dans les nœuds (distincts)* 0.1 *et* 1.2*).* □

La notion de *XDF* triviale (c'est-à-dire toujours satisfaite, pour tout arbre t contenant des instances de ses chemins) a été étudiée dans (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a), d'où nous pouvons synthétiser la propriété suivante :

Proposition 3.1 - Dépendance triviale - Une *XDF* $(C, (\{P_1\,[E_1], \ldots, P_k\,[E_k]\} \rightarrow Q\,[E]))$, où $P_1, \ldots, P_k, Q$ sont des chemins simples, est une *XDF triviale* si l'une des conditions suivantes est respectée :

(*i*) $Q = P_i$ et $E = E_i = N$ pour tout $1 \leq i \leq k$. Cas où un chemin dans la partie gauche est identique au chemin dans la partie droite, avec l'égalité de nœud.

(*ii*) $P_i = Q/X$, $Last(X) \in \Sigma_{ele}$ pour un i, $1 \leq i \leq k$ et $E_i = N$. Dans ce cas, il existe un chemin P_i dans la partie gauche dont Q est le préfixe. Ce chemin P_i finit dans un élément (et non pas dans un nœud *data* ou attribut), et son option d'égalité est l'identité de nœuds.

(*iii*) $Last(Q) \in \Sigma_{att}$, $E = V$, $P_i = Parent(Q)$ et $E_i = N$ pour un i, $1 \leq i \leq k$. C'est le cas où un nœud (père) détermine la valeur de son attribut. □

On remarque que pour le document de la figure 1, XDF_8 : ({ $bd/projet/pnom$ $[N]$ } $\rightarrow$ $bd/projet$) est toujours satisfaite. De même, XDF_9 : ($bd/projet$, ({ $/fournisseur$ $[N]$} $\rightarrow$ $/fournisseur/@fnom$)) est une dépendance triviale.

3.2. *Informations incomplètes*

Dans le contexte relationnel, le problème d'étendre les contraintes d'intégrité pour prendre en considération la présence des valeurs nulles a été beaucoup étudié, en particulier par (Codd, 1979, Lipski, 1981, Lien, 1982, Imieliński *et al.*, 1984, Atzeni *et al.*, 1986, Imielinski, 1989, Levene *et al.*, 1999). Une relation incomplète nécessite de considérer l'ensemble de toutes les relations pouvant être obtenues en remplaçant les occurrences de valeurs nulles (*null*) par des valeurs du domaine des attributs de la relation. Pour étendre la satisfaction d'une dépendance fonctionnelle au cas où les relations sont incomplètes, deux notions ont été introduites : la *satisfaction forte* et la *satisfaction faible*.

En suivant les idées proposées pour le modèle relationnel, les deux types de satisfaction peuvent être également envisagées dans le contexte XML :

– *Satisfaction forte* : la *XDF* doit être satisfaite dans tous les mondes possibles. Un document incomplet satisfait la *XDF* si et seulement si aucun remplacement d'une valeur *null* par une valeur différente de *null* ne remet en cause sa validité ;

– *Satisfaction faible* : la *XDF* doit être satisfaite dans au moins un monde. Dans ce cas, lorsqu'une valeur *null* est remplacée par une valeur différente de *null*, une nouvelle validation est nécessaire.

Si nous considérons des données incomplètes, la construction des n-uplets d'instance de motif selon la définition 3.2 est légèrement différente car, pour chaque instance de chemin incomplète, une valeur *null* est ajoutée au n-uplet. Soit un motif M formé à partir de $\{C/\vec{P},\ C/Q\}$ avec $\vec{P} = \{P_1, \ldots, P_k\}$, et soient des instances incomplètes de M (notées $\mathcal{I}$) : nous construisons des n-uplets $\mathcal{I}[C/\vec{P},\ C/Q]$ selon la procédure de la définition 3.2, ces n-uplets contenant soit des valeurs de nœuds, soit des valeurs inconnues (notées $\perp$) quand l'instance de P_i ($i \in [1 \ldots k]$) ou de Q sur $\mathcal{I}$ est incomplète.

La satisfaction faible des *XDF* peut être traitée comme dans le modèle relationnel, en considérant que les valeurs inconnues dans les n-uplets sont différentes les unes des autres. Une fois que nous avons les n-uplets d'instances de motif, nous appliquons un algorithme *chase* (Maier *et al.*, 1979, Levene *et al.*, 1999). Le principe de l'algorithme consiste à exécuter la boucle suivante, dont nous expliquons les concepts dans la suite. R est une instance de relation composée de tous les n-uplets $\mathcal{I}[C/\vec{P},\ C/Q]$, qui contiennent éventuellement des valeurs inconnues.

Tant que il existe $\mathcal{I}_1, \mathcal{I}_2 \in R$ et
il existe une *XDF* $(C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow Q\ [E]))$ avec $E_i = V$, $E = V$, $\vec{P} = \{P_i\}$, $(i \in [1 \ldots k])$ telle que
$\mathcal{I}_1[C/\vec{P}] = \mathcal{I}_2[C/\vec{P}]$ et $\neg(\mathcal{I}_1[C/Q] \cong \mathcal{I}_2[C/Q])$ **faire**
$\mathcal{I}_1[C/Q], \mathcal{I}_2[C/Q] := lub(\mathcal{I}_1[C/Q], \mathcal{I}_2[C/Q])$

Dans cet algorithme, les concepts de base sont les suivants :

1) On considère un domaine **EV** contenant les valeurs (ou données) possibles dans T (**V** dans la définition 2.2), plus des valeurs inconnues. Chaque occurrence de valeur inconnue représente une valeur potentiellement différente (on les distingue donc en les notant $\perp$ avec des indices).

2) Étant données deux valeurs v_i et v_j dans **EV**, v_i est équivalent en information (*information-wise equivalent*) à v_j ($v_i \cong v_j$), ssi v_i et v_j sont identiques syntaxiquement, c'est-à-dire qu'ils ont les mêmes noms. Si v_i n'est pas équivalent en information à v_j, on écrit $\neg(v_i \cong v_j)$. Par exemple, $\perp_2 \cong \perp_2$ et *iris* $\cong$ *iris*, mais $\neg$(*iris* $\cong$ *hilary*), $\neg(\perp_2 \cong \perp_1)$.

3) L'égalité utilisée dans le *chase* est une égalité à trois valeurs (vraie, fausse et inconnue). Étant données deux valeurs v_i et v_j dans **EV**, l'égalité à trois valeurs est définie selon les cas ci-dessous :

a) Si v_i et v_j sont des valeurs différentes de *null* alors $v_i = v_j$ est évaluée comme vraie si $v_i \cong v_j$; sinon elle est évaluée comme fausse.

b) $\perp_j = \perp_i$ est évaluée comme vraie si $i = j$, sinon elle est évaluée comme inconnue.

c) Si v_i est une valeur *null* et v_j est une valeur différente de *null*, alors $v_i = v_j$ est évaluée comme inconnue.

4) Une relation d'ordre partiel ($\sqsubseteq$) est définie sur les valeurs. Les valeurs différentes de *null* sont incomparables. De plus nous avons $\perp \sqsubseteq$ *nonnull values* $\sqsubseteq$ *inc* où *inc* est un type de valeur *null* qui indique une incohérence.

5) *Least upper bound operator* (lub) : la plus petite borne supérieure de deux valeurs v_1 et v_2 dans **EV** est définie de la façon suivante :

a) Si $v_1 \sqsubseteq v_2$ alors $lub(v_1, v_2) = v_2$; sinon

b) Si $v_2 \sqsubseteq v_1$ alors $lub(v_1, v_2) = v_1$; sinon

c) Si $\neg(v_2 \sqsubseteq v_1)$ et $\neg(v_1 \sqsubseteq v_2)$ alors $lub(v_1, v_2) = inc$.

L'instance de relation R satisfait faiblement une *XDF* si et seulement si la procédure *chase* est cohérente (aucune valeur *inc* n'est introduite par la boucle). La définition suivante formalise cette notion dans le cadre XML.

Définition 3.4 - Satisfaction de *XDF* : cas d'informations incomplètes - Soient $T = (t, type, value)$ un document XML et $\gamma = (C, (\{P_1\ [E_1], \ldots, P_k\ [E_k]\} \rightarrow Q\ [E]))$ une *XDF*. Soit M le motif formé à partir de l'ensemble $\{C/\vec{P}, C/Q\}$, où $\vec{P} = P_1, \ldots, P_k$. Soit R l'instance de relation composée par tous les n-uplets d'instances $\mathcal{I}[C/\vec{P}, C/Q]$ du motif M dans T (éventuellement avec des valeurs inconnues, distinguées), les instances coïncidant au moins sur le chemin C. Nous disons que T satisfait *faiblement* γ (et cela s'écrit $T \models_{faible} \gamma$) si et seulement si la procédure $chase(R, \gamma)$ est cohérente. □

Par exemple, considérons le document T de la figure 2 et la dépendance XDF_3. Nous voulons vérifier si $T \models_{faible} XDF_3$, sachant qu'il manque des informations dans T. Nous considérons donc M le motif formé à partir des chemins dans XDF_3, à savoir :
$C/P_1 = bd/projet/fournisseur/@fnom$,
$C/P_2 = bd/projet/fournisseur/composant/@cnom$ et
$C/Q = bd/projet/fournisseur/composant/prix$.

A partir du motif M nous obtenons deux instances dans T, $\mathcal{I}^1$ et $\mathcal{I}^2$ qui coïncident en C, et par conséquent les deux n-uplets suivants :

$$\mathcal{I}^1[C/P_1, C/P_2, C/Q] = [MSI, 955XNeo, 185{,}00]$$
$$\mathcal{I}^2[C/P_1, C/P_2, C/Q] = [MSI, 955XNeo, \perp_1]$$

L'instance de relation R contient ces deux n-uplets. La procédure *chase* appliquée à R va remplacer la valeur $\perp_1$ dans $\mathcal{I}^2[C/Q]$ par la valeur 185,00,

donc aucune valeur *inc* n'est introduite. Ainsi, la procédure est cohérente et $T \models_{faible} XDF_3$ selon la définition 3.4. Il existe au moins une valeur pour laquelle XDF_3 est satisfaite.

Remarquons qu'il est aussi possible de choisir une satisfaction forte des *XDF*. Dans ce cas, les valeurs inconnues ne sont pas distinguées (notées $\perp$ sans indice) dans les tuples qui composent R et on applique l'algorithme *SatFort* sur R (Levene *et al.*, 1999).

3.3. *Dépendances d'inclusion*

Dans le cadre du modèle relationnel, les dépendances d'inclusion (*DI*) formalisent des contraintes sur plusieurs relations (Mitchell, 1983, Casanova *et al.*, 1982). Une *DI* énonce des conditions concernant l'occurrence de valeurs dans une relation par rapport aux valeurs dans une autre. C'est une contrainte entre relations dont un cas particulier est la contrainte référentielle (contrainte de clef étrangère). Des contraintes d'inclusion dans le contexte XML (*XDI*) ont été définies dans (Fan *et al.*, 2002, Buneman *et al.*, 2001b, Vincent *et al.*, 2004b). Dans la définition suivante, la *XDI* est exprimée à partir de chemins, comme dans les propositions citées précédemment, mais les chemins ne sont pas liés à un schéma donné. En outre, la *XDI* peut concerner une partie spécifique du document T, partie représentée par le contexte dans notre définition.

Définition 3.5 - Dépendance d'inclusion - Une dépendance d'inclusion *XDI* a la forme $(C, (\{P_1, \ldots, P_k\} \subseteq \{Q_1, \ldots, Q_k\}))$, où pour tout i, $i \in [1, k]$, C/P_i et C/Q_i sont des chemins. Le chemin C représente le contexte dans lequel la dépendance d'inclusion doit être respectée. □

Lorsque $C = []$ il peut être omis et la dépendance d'inclusion s'applique dans tout l'arbre. Une dépendance d'inclusion peut être définie sur les nœuds internes ou les nœuds feuilles d'un arbre t, mais dans les deux cas l'égalité par valeurs est utilisée. La définition suivante précise l'interprétation d'une dépendance d'inclusion, que le document soit complet ou pas.

Définition 3.6 - Satisfaction de XDI - Soient T un arbre XML, $\varphi = (C, (\{P_1, \ldots, P_k\} \subseteq \{Q_1, \ldots, Q_k\}))$ une *XDI*, M le motif formé à partir de $\{C/P_1, \ldots, C/P_k\}$ et N le motif formé à partir de $\{C/Q_1, \ldots, C/Q_k\}$. Nous disons que T satisfait φ (et cela s'écrit $T \models \varphi$) si et seulement si pour toute instance $\mathcal{I}$ (éventuellement incomplète) du motif M dans T, il existe une instance $\mathcal{J}$ du motif N dans T qui coïncide avec $\mathcal{I}$ au moins sur C et telle que : $\forall i \in [1, \ldots, k]$, $\mathcal{I}[C/P_i] \sqsubseteq \mathcal{J}[C/Q_i]$ où le symbole $\sqsubseteq$ dénote la relation d'ordre partiel introduite dans la section 3.1 ($\perp \sqsubseteq$ *nonnull values* $\sqsubseteq$ *inc*) qui prend en compte l'égalité multivaluée. □

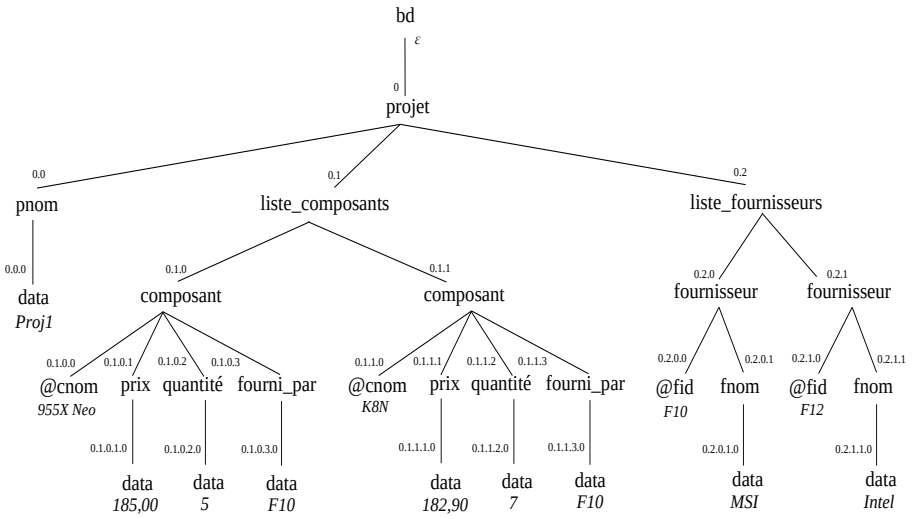

Figure 3. *Les composants sont liés à leur fournisseur (via l'élément "fourni_par").*

L'arbre de la figure 3 décrit un projet contenant une liste de composants et une liste de fournisseurs. Nous voulons exprimer la contrainte suivante : sous un projet, chaque fois qu'une valeur apparaît dans le chemin *liste_composants/composant/fourni_par*, elle doit obligatoirement apparaître dans *liste_fournisseurs/fournisseur/@fid*. En d'autres termes, le fournisseur de chaque composant doit être inclus dans la liste de fournisseurs d'un projet. Cette dépendance d'inclusion est écrite de la façon suivante :

$$XDI_1 : (bd/projet, (\{/liste_composants/composant/fourni_par\} \subseteq \{/liste_fournisseurs/fournisseur/@fid\}))$$

La dépendance XDI_1 est vérifiée par le document T de la figure 3 car pour les deux instances du motif contenant *bd/projet/liste_composants/composant/fourni_par* le n-uplet vaut [$F10$] et le n-uplet [$F10$] existe pour une instance du motif contenant *bd/projet/liste_fournisseurs/fournisseur/@fid*.

La vérification d'une dépendance d'inclusion en cas d'information incomplète utilise une satisfaction faible ($\sqsubseteq$ permet de comparer les valeurs *null* et les valeurs différentes de *null*). Le document XML T de la figure 4 illustre un projet avec une inconnue : le fournisseur (*fourni_par*) du composant de position 0.1.1. Cet arbre vérifie néanmoins la dépendance d'inclusion XDI_1, y compris pour l'instance incomplète $I = \varepsilon/0/0.1/0.1.1$ du chemin *bd/projet/liste_composants/composant/fourni_par*.

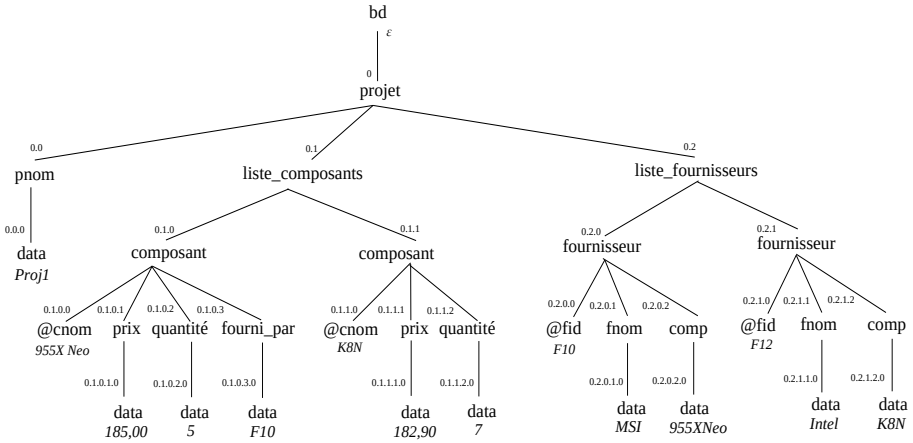

Figure 4. *Composants et fournisseurs peuvent être liés par une dépendance de lien inverse.*

Une contrainte particulière, correspondant à une double dépendance d'inclusion, a été étudiée dans (Buneman *et al.,* 2001b, Fan *et al.,* 2003) pour les données XML. Elle est dénommée dépendance de lien inverse, ou *XDInv*.

Par exemple, considérons la *XDInv* suivante sur l'arbre de la figure 4 :

$$\begin{aligned} XDInv_1 : (\, bd/projet, (\, \{\, & /liste_composants/composant/@cnom, \\ & /liste_composants/composant/fourni_par\} = \\ \{\, & /liste_fournisseurs/fournisseur/comp, \\ & /liste_fournisseurs/fournisseur/@fid\ \}\,)) \end{aligned}$$

Dans un même projet, chaque n-uplet formé du nom d'un composant et de son fournisseur trouvé dans la liste de composants doit être aussi obtenu dans la liste de fournisseurs, et vice-versa.

Une dépendance de lien inverse permet notamment de représenter la sémantique des données XML obtenues à partir de bases de données orientées objet (Cattell, 2000), pour lesquelles les liens de composition inverse sont une partie fondamentale de l'information.

Dans (Fan *et al.,* 2003) les dépendances d'inclusion et de lien inverse sont définies en termes de chemins simples (séquences d'étiquettes) et sont liées aux attributs ID et IDREF d'une DTD. Dans (Buneman *et al.,* 2001b), les chemins de la dépendance sont plus complexes, et peuvent inclure des symboles de concaténation, union et fermeture de Kleene. Nous proposons ci-dessous d'exprimer ces dépendances dans le même formalisme que les *XDF* et *XDI*.

Définition 3.7 - Dépendance de lien inverse *XDInv* **-** Une dépendance de lien inverse a la forme $(C, (\{P_1, \ldots, P_k\} = \{Q_1, \ldots, Q_k\}))$ où le chemin C représente le

contexte dans lequel la dépendance de lien inverse doit être respectée, et P_1, ..., P_k, Q_1, ..., Q_k sont des chemins. □

Une dépendance de lien inverse est satisfaite si dans le contexte C, l'ensemble de valeurs trouvées en suivant les chemins P_1, ..., P_k est identique à l'ensemble de valeurs obtenues en suivant $Q_1, \ldots, Q_k$. Les informations incomplètes peuvent ici aussi être traitées par une satisfaction faible ($\sqsubseteq$ permettant de comparer les valeurs *null* et les valeurs différentes de *null*).

Définition 3.8 - Satisfaction de dépendance de lien inverse - Soient T un arbre XML, une *XDInv* $\eta = (C, (\{P_1, \ldots, P_k\} = \{Q_1, \ldots, Q_k\}))$, et soient M le motif formé à partir de $\{C/P_1, \ldots, C/P_k\}$ et N le motif formé à partir de $\{C/Q_1, \ldots, C/Q_k\}$. Nous disons que T satisfait η (et cela s'écrit $T \models \eta$) si et seulement si pour toute instance $\mathcal{I}$ (éventuellement incomplète) du motif M dans T, il existe une instance $\mathcal{J}$ du motif N dans T qui coïncide avec $\mathcal{I}$ au moins sur C et telle que : $\mathcal{I}[C/P_1, \ldots, C/P_k] =_V \mathcal{J}[C/Q_1, \ldots, C/Q_k]$; et inversement pour toute instance $\mathcal{J}$ du motif N dans T, il existe une instance $\mathcal{I}$ du motif M dans T qui coïncide avec $\mathcal{J}$ au moins sur C et telle que : $\mathcal{I}[C/P_1, \ldots, C/P_k] =_V \mathcal{J}[C/Q_1, \ldots, C/Q_k]$. □

Nous remarquons que la plupart des exemples de triggers décrits dans (Bonifati *et al.*, 2001, Shao *et al.*, 2005) ont justement pour vocation de maintenir ce type de contraintes (sans les citer). Un environnement dans lequel il serait possible de simplement spécifier des contraintes et qui prendrait en charge leur maintenance automatique lors de mises à jour représente une alternative à la définition de triggers dans le cadre XML.

4. Définition et vérification de clés et de clés étrangères

4.1. *Clés et clés étrangères*

Les clés et les clés étrangères sont des cas particuliers de dépendances, les plus couramment implantés dans les systèmes de gestion de bases de données. Pour XML, à notre connaissance seul le langage de schéma *XML Schema* (Thompson *et al.*, 2006) permet de définir des clés et des clés étrangères. Il est important de noter que les définitions de schémas et de contraintes d'intégrité peuvent interférer et être globalement incohérentes, nous revenons sur ce sujet en section 5. Une clé pour XML peut également être définie indépendamment d'un schéma (Buneman *et al.*, 2001a, Buneman *et al.*, 2003b), moyennant tout de même l'hypothèse que le document soit conforme au motif défini par la clé. Nous reprenons cette proposition, qui décrit une clé en trois parties : (*i*) le contexte dans lequel la clé doit être vérifiée, (*ii*) l'élément identifié par la clé, et (*iii*) les parties (valuées) de l'élément qui forment la clé.

Définition 4.1 - Contrainte de clé (Buneman *et al.*, 2001a) - Une clé ϕ est représentée par une expression de chemin de la forme : $\phi = (C, (CI, \{P_1, \ldots, P_k\}))$ où C, CI et P_i sont des chemins. Le chemin C est appelé le *chemin contexte*, CI est le *chemin cible* et $P_i, \ldots, P_k$ sont appelés *chemins clés* de γ. □

Lorsque $C = []$, γ est une *clé absolue*, et on peut abréger l'expression à $(CI, \{P_1, \ldots, P_k\})$. Par exemple, dans la figure 1, nous pouvons considérer les clés $K_1 : (bd/projet, \{/pnom/data\})$ et $K_2 : (bd/projet, (/fournisseur, \{/@fnom\}))$. K_1 est une clé absolue qui établit que chaque projet du document est identifié de façon unique par la valeur de *pnom/data*. K_2 est une clé relative : chaque fournisseur *d'un même projet* est identifié par la valeur de $@fnom$.

La satisfaction d'une clé peut être définie de façon intuitive : s'il y a, dans le même contexte, deux ensembles de valeurs déterminant un élément cible qui sont identiques, alors il ne peut s'agir que du même élément cible.

De fait, les clés sont un cas particulier de dépendances fonctionnelles, où Q (la partie droite de la XDF) est préfixe de tous les chemins $P_1, \ldots, P_k$ et $E_1, \ldots, E_k$ représentent tous l'égalité de valeurs (V) alors que E indique l'identité de nœuds (N). De plus les clés exigent la présence des valeurs (aucune valeur $\perp$) et l'unicité des chemins composant la clé, ce qu'exprime la définition suivante.

Définition 4.2 - Satisfaction de clé - Soient $\phi = (C, (CI, \{P_1, \ldots, P_k\}))$ une clé, un document $T = (t, type, value)$ respecte ϕ, noté $T \models \phi$, si et seulement si T satisfait la *XDF* $\varphi = (C, (\{CI/P_1, \ldots, CI/P_k\} \rightarrow CI\ [N]))$ et il y a une et une seule instance de chaque chemin P_i sous chaque nœud dans $n[\![CI]\!]_t$, pour tout $n \in \epsilon[\![C]\!]_t$. La notation $p[\![P]\!]_t$ représente l'ensemble de positions auxquelles on aboutit en suivant le chemin P à partir de la position p dans t. □

Ainsi, si on considère les deux spécifications suivantes :

$$\varphi_1 = (bd, (\{/projet/pnom\} \rightarrow /projet\ [N]))$$
$$\phi_1 = (bd, (/projet, \{/pnom\}))$$

Nous pouvons remarquer qu'elles ne décrivent pas la même contrainte, vu que φ_1 est moins restrictive. Le document T de la figure 5 n'est pas valide par rapport à ϕ_1 vu qu'il y a deux noms de projet sous le nœud *projet* de position 2. Par contre la contrainte φ_1 est respectée par T.

Une clé étrangère détermine une référence entre valeurs. Dans le contexte XML, elle est donc composée de deux ensembles de chemins : ceux de la référence et ceux de la clé référencée, précisés dans la définition suivante.

Définition 4.3 - Contrainte de clef étrangère XML (Abrão *et al.*, 2004) - Une clé étrangère δ est représentée par une expression de la forme : $(C, (CI^f,$

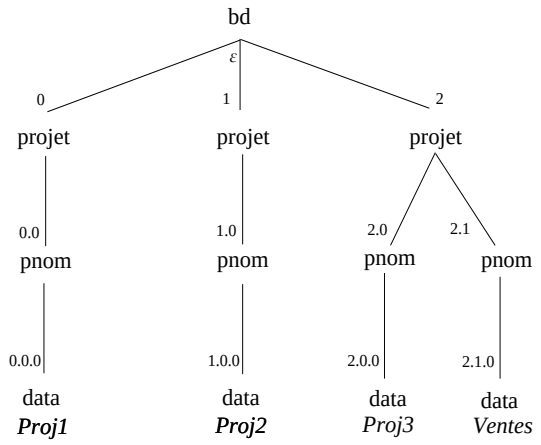

Figure 5. *Document XML qui respecte φ_1 mais ne respecte pas ϕ_1.*

$\{P_1^f, \ldots, P_k^f\}) \subseteq (CI, \{P_1, \ldots, P_k\}))$ où $P_1^f, \ldots, P_k^f$ sont appelés *chemins clé étrangère* et $K = (C, (CI, \{P_1, \ldots, P_k\}))$ est une clé. □

En considérant le document T représenté dans la figure 3, nous pouvons écrire une clef étrangère pour déterminer que chaque composant d'un projet a pour fournisseur, un fournisseur inclus dans la liste de fournisseurs. Pour exprimer cette contrainte, la clef étrangère FK_2 s'écrit :

$$K_1 = (/bd/projet, (/liste_fournisseurs/fournisseur, \{/@fid\}))$$
$$FK_2 = (/bd/projet, (/liste_composants/composant, \{/fourni_par\}) \subseteq (/liste_fournisseurs/fournisseur, \{/@fid\}))$$

Une clé étrangère $(C, (CI^f, \{P_1^f, \ldots, P_k^f\}) \subseteq (CI, \{P_1, \ldots, P_k\}))$ peut être décrite par la *XDI* suivante : $(C, (\{CI^f/P_1^f, \ldots, CI^f/P_k^f\} \subseteq \{CI/P_1, \ldots, CI/P_k\}))$. Mais la *XDI* généralise la clé étrangère puisque les valeurs trouvées en suivant les chemins dans la partie gauche peuvent référencer un ensemble de valeurs qui ne forment pas une clé, donc pas forcément uniques.

4.2. *Vérification des clés*

Des méthodes ont été proposées pour vérifier automatiquement des clés en XML (Chen *et al.*, 2002b, Bouchou *et al.*, 2003, Abrão *et al.*, 2004, Bouchou *et al.*, 2007, Lima, 2007). Il s'agit essentiellement de pouvoir :

– Déterminer les nœuds concernés par la clé : nœuds *contexte*, nœuds *cible* et nœuds formant la clé.

– Remonter les valeurs qui forment la clé, qui se situent aux feuilles de l'arbre du document XML, vers leur contexte afin de les tester à ce niveau. Pour une clé il faut vérifier que dans chaque contexte les n-uplets de valeurs sont tous distincts.

Pour réaliser la première étape, qui est de localiser les parties du document concernées par la clé, l'outil naturel est l'automate à états finis dans la mesure où il s'agit de suivre les chemins qui définissent la clé, depuis la racine. Comme indiqué dans la section 2.1, on utilise 3 automates finis pour représenter une clé, le premier pour le chemin de la racine au contexte (noté A), le deuxième pour le chemin du contexte à la cible (noté A'), et le troisième pour les chemins de la cible à chaque élément du n-uplet clé (noté A''). La figure 6 illustre les 3 automates associés à la clé K_2 : $(bd/projet, (/fournisseur, \{/@fnom\}))$.

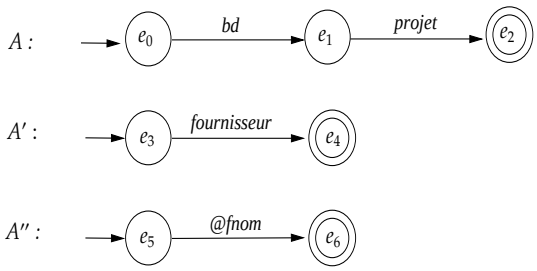

Figure 6. *Automates A, A′ et A″ associés à la clé K_2.*

Pendant un parcours linéaire de l'arbre dans l'ordre du document (c'est-à-dire en profondeur d'abord), on associe à chaque nœud une configuration d'automate, c'est-à-dire le nom (A, A' ou A'') de l'automate correspondant et l'état dans lequel il est à ce nœud. Cette affectation se fait de la façon suivante : on commence avec l'automate A dans son état initial, si l'étiquette de la racine est celle de la première transition de A alors on associe $A.e_1$ à la racine, et pour chaque étiquette de nœud rencontrée on teste les transitions de l'automate. Lorsqu'on arrive dans un état final de A, on teste la prochaine étiquette par rapport à la première transition de A'. De même, après un état final de A', on utilise les transitions de A''.

Les configurations reflètent le rôle des nœuds par rapport à la clé : les nœuds contextes ont pour configuration $A.f_A$ où f_A est l'état final de A, les nœuds cibles ont pour configuration $A'.f_{A'}$ où $f_{A'}$ est l'état final de A' et les nœuds clés ont pour configuration $A''.f_{A''}$ où $f_{A''}$ est l'état final de A''. De plus, les noeuds se trouvant sur les chemins ont une configuration associée, ce qui permet de faire transiter les valeurs à tester depuis les feuilles vers les nœuds contextes. C'est le fait de rassembler les valeurs depuis les feuilles vers les cibles, puis des cibles vers les contextes, qui permet de réaliser correctement les n-uplets d'instances de motifs.

Dans (Bouchou *et al.*, 2007, Lima, 2007), nous montrons comment associer aux contraintes une grammaire attribuée, qui représente le processus de vérification. Une grammaire attribuée est une grammaire hors contexte étendue par des attributs (Aho *et al.*, 1988). Chaque symbole de la grammaire est lié à un ensemble d'attributs fini, éventuellement vide. Chaque attribut a un domaine de valeurs. La valeur de chaque attribut à chaque nœud de l'arbre syntaxique est définie par des règles sémantiques. Les attributs associés à un symbole sont de deux types :

– Synthétisés : la règle sémantique utilise les valeurs des attributs associés aux fils pour donner une valeur à l'attribut du nœud courant.

– Hérités : la règle sémantique utilise les valeurs des attributs associés aux frères gauches et au père pour donner une valeur à l'attribut du nœud courant.

Nous utilisons une grammaire générique de 3 règles pour représenter tout document T, en considérant par ailleurs que tous les éléments de l'alphabet Σ de T, sauf *data*, sont liés à un ensemble d'attributs $\{conf, c, ci, k\}$ pour chaque clé, attributs dont nous décrivons le rôle ci-après. Pour simplifier, nous utilisons les trois règles de la grammaire sous la forme suivante :

– pour un *nœud racine* (a des fils mais pas de père) : $Root \rightarrow \alpha_1 \ldots \alpha_m$

– pour *un nœud interne* (a un père et des fils) : $R \rightarrow \alpha_1 \ldots \alpha_m$

– pour *un nœud "feuille"* (père de *data* ou attribut) : $R \rightarrow data$

Les valeurs des attributs hérités et des attributs synthétisés sont définies par les règles sémantiques associées aux règles de la grammaire et calculées au cours du parcours du document. Pour chaque clé K, nous utilisons un attribut hérité, nommé $conf$, pour lier chaque nœud concerné par la clé à une configuration (comme expliqué précédemment). Ainsi l'attribut hérité $conf$ dénote le rôle du nœud par rapport à K. Par exemple, nous montrons dans la table 1 la règle sémantique qui permet de calculer la valeur de $conf$ aux nœuds fils de la racine (règle pour la racine). On y voit que pour le nœud racine, $conf$ est la configuration de l'automate du chemin contexte qui, partant de son état initial, reconnaît la première étiquette de ce chemin. Dans l'exemple 4.1, $conf = \{A_2.e_1\}$. On voit également dans la table 1 que pour chaque fils, on essaie d'avancer dans l'automate du chemin contexte, ou bien de passer à l'automate du chemin cible dans le cas où la clé est une clé absolue (le chemin contexte est alors vide).

L'ensemble des règles sémantiques qui déterminent la valeur de $conf$ à chaque nœud est détaillé dans (Bouchou *et al.*, 2007, Lima, 2007), nous en décrivons simplement le fonctionnement dans l'exemple 4.1.

Exemple 4.1 - *Considérons le document T de la figure 1 et la clé K_2 :* $(bd/projet, (/fournisseur, \{/@fnom\}))$ *avec les automates finis correspondants (figure 6, respectivement A_2, A_2' et A_2''). Comme le montre la figure 7, on associera au nœud de la*

Tableau 1. *Règle sémantique pour l'attribut conf (Production de la racine).*

Production	Attributs

$ROOT \rightarrow \alpha_1 \ldots \alpha_m$	$ROOT.conf := \{ A.q_1 \mid \delta_A (q_0, ROOT) = q_1 \}$
	
	/* *Attributs hérités* */
	for each α_i $(1 \leq i \leq m)$ do
	$\alpha_i.conf := \{ A.q' \mid \delta_A (q_1, \alpha_i) = q' \}$
	if $(q_1 \in F_A)$
	then $\alpha_i.conf := \alpha_i.conf \cup \{ A'.q'_1 \mid \delta_{A'}(q'_0, \alpha_i) = q'_1 \}$
	end for

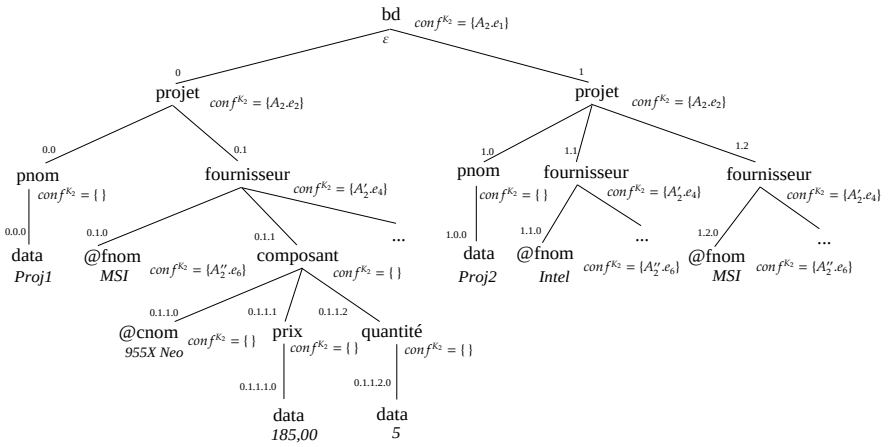

Figure 7. *Attribut hérité conf pour K_2.*

racine la configuration $\{A_2.e_1\}$. Pour calculer $conf^{K_2}$ pour le nœud projet à la position 0, nous exécutons la transition dans A_2 d'étiquette ***projet****. Le résultat est $\{A_2.e_2\}$. Comme il s'agit d'un état final pour A_2, nous passons à l'automate A'_2 pour la suite. Ainsi pour traiter le nœud 0.1, nous exécutons la transition d'étiquette* ***fournisseur*** *et nous obtenons $\{A'_2.e_4\}$. Pareillement, l'ensemble $\{A''_2.e_6\}$ est affecté au nœud 0.1.0. Un ensemble vide indique les nœuds non concernés par la clé, c'est le cas par exemple du nœud 0.0. A la fin du parcours du document XML, chaque nœud a une valeur pour son attribut $conf^{K_2}$.* □

Une fois tous les fils traités, le nœud père est reconsidéré (à la rencontre de la balise fermante pendant le parcours du document). Toute valeur associée à la

Tableau 2. *Règle sémantique pour l'attribut k (Production des nœuds data).*

Production	Attributs

$R \rightarrow data$	/* *Attributs synthétisés* */
	for each configuration $\overline{A}.q$ in $R.conf$ do
	if $(\overline{A} = A'') \wedge (q \in F_{A''})$
	then $R.k := < value(data) >$
	end for

clé doit être remontée, d'abord jusqu'à l'élément identifié (la cible), où s'effectue un regroupement en n-uplet, puis le n-uplet est remonté au contexte pour la vérification de l'unicité. Enfin les résultats des vérifications sont remontés à la racine, donnant la décision pour le document complet. Les *attributs synthétisés* servent à ce processus : un attribut k est utilisé entre une feuille et un nœud cible, un attribut ci est utilisé entre un nœud cible et un nœud contexte, et un attribut c est utilisé entre un nœud contexte et la racine.

Les attributs c, ci et k de chaque nœud reçoivent des valeurs selon le rôle que le nœud joue par rapport à la clé, donné par la valeur de son attribut $conf$, et aussi selon les valeurs des attributs c, ci et k des nœuds fils. Par exemple, la règle de la table 2 correspond à la règle de production pour des nœuds *data* : l'attribut k du père obtient la valeur des données s'il est un nœud clé pour la contrainte ($conf$ correspond à l'état final de l'automate clé). L'exemple 4.2 montre les valeurs que prennent les attributs c, ci et k au fur et à mesure du parcours du document.

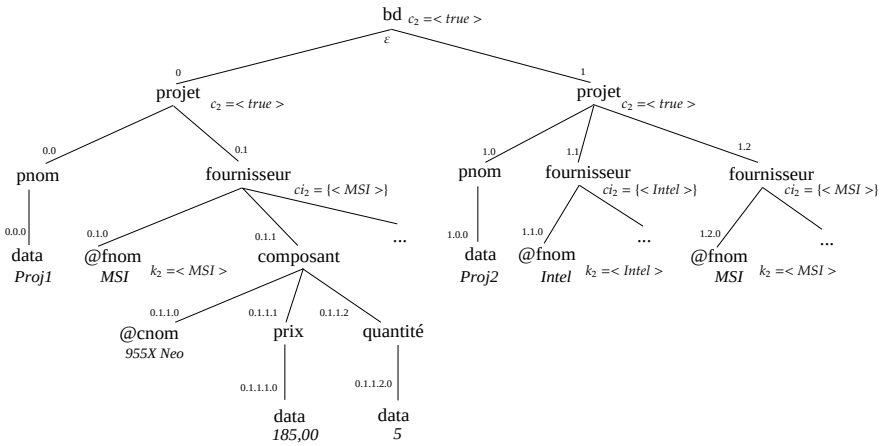

Figure 8. *Attributs synthétisés c,ci et k pour K_2.*

Exemple 4.2 - *La figure 8 illustre le calcul des valeurs des attributs synthétisés pour K_2. Les valeurs des données pour K_2, obtenues à partir des nœuds étiquetés* ***@fnom*** *sont accumulées en chaque attribut k_2. Les valeurs clés doivent être toutes présentes et regroupées (si la clé est composée) quand le nœud* ***fournisseur*** *(cible) est atteint. Au nœud contexte (****projet****), comme tous les n-uplets récupérés des divers ci_2 sont distincts (et complets), l'attribut c_2 est affecté avec la valeur true. Cela veut dire que pour les deux nœuds* ***projet****, K_2 est respectée. A la racine, l'attribut c_2 est affecté avec la valeur true, indiquant que le document satisfait la clé K_2 pour tous les contextes.* □

Une vérification de clé se réalise donc en un parcours linéaire du document. Pendant ce parcours, nous construisons un index des valeurs des clés, appelé *keyTree*, utile pour vérifier les clés étrangères et pour réaliser une vérification incrémentale en cas de mise à jour du document.

5. Travaux liés

5.1. *Vérifications de contraintes*

Dans (Lima, 2007), il est montré comment les principes de vérification qui viennent d'être esquissés pour les clés se généralisent à un nombre quelconque de dépendances, exprimées via un ensemble de chemins. Dans (Chen *et al.*, 2002b), les auteurs présentent les principes d'une validation incrémentale, de complexité linéaire sur la taille du document, mais qui ne s'applique qu'à des clés et ne traite qu'une mise à jour à la fois. On trouve également dans (Bouchou *et al.*, 2007, Lima, 2007) une description de la vérification incrémentale des clés ainsi que le détail des résultats expérimentaux obtenus. L'ordre de grandeur se situe aux alentours de la seconde pour de petits documents et peu de contraintes, et jusqu'à 2 minutes pour des documents d'un million de nœuds et pour une dizaine de contraintes. Il est à noter que, plus que le nombre de contraintes, c'est le nombre d'instances des contraintes dans le document qui influe sur les temps d'exécution.

Certaines propositions ont été faites pour la vérification des contraintes XML, qui utilisent la logique. La proposition de (Benedikt *et al.*, 2002) consiste à traduire les contraintes en formules logiques du premier ordre et à faire correspondre aux mises à jour des documents des calculs sur ces formules. Ils utilisent des informations tirées du schéma des documents pour optimiser ces calculs.

D'autres travaux utilisent la logique modale pour modéliser les données semi-structurées puisque ces dernières peuvent être vues comme des graphes ayant des arcs étiquetés - et donc comme des modèles de Kripke. Dans (Demri, 2003), l'auteur montre que différents problèmes peuvent être considérés selon cette optique. Dans ce cadre, nous pouvons citer (Alechina, 1997) pour les

contraintes de chemins et (Thion, 2004, Bidoit *et al.*, 2004, Bidoit *et al.*, 2006, Bidoit *et al.*, 2007) pour les contraintes de références "bien typées".

5.2. *XML Schema*

Parmi les contraintes d'intégrité prises en charge par les systèmes de gestion de bases de données relationnels il y a également les *contraintes de domaine*, qui permettent de spécifier les valeurs permises pour un attribut donné. Dans (Lima, 2007) l'auteur propose un formalisme proche de celui décrit dans cet article pour de telles contraintes. Le langage de définition de schéma XML Schema (Thompson *et al.*, 2006) permet également de définir très précisément les domaines de valeurs permis pour les données contenues dans un document XML. De plus, les clés et clés étrangères qu'il permet de spécifier sont comparables à celles présentées en section 4 : elles peuvent être absolues ou relatives à un contexte. Concernant les algorithmes qui implémentent la vérification des clés et clés étrangères définies en XML Schema nous manquons de sources qui permettraient une comparaison.

Dans (Benedikt *et al.*, 2003), les auteurs étudient plusieurs langages de chemins et en particulier X_r, le sous-langage de XPath qui permet la navigation dans la direction d'ancêtre et de descendant, mais ne permet pas la navigation horizontale ou l'utilisation des prédicats, c'est-à-dire que X_r revient à notre langage *PL*. Ces auteurs montrent que X_r est exactement le sous-langage permis dans XML Schema pour la définition des clés.

Dans (Arenas *et al.*, 2002a, Fan *et al.*, 2001, Kot *et al.*, 2007) les auteurs étudient les interactions qui surviennent entre des contraintes de structure (le schéma) et des contraintes d'intégrité. Dans nos travaux nous avons considéré les clés et les clés étrangères en suivant l'approche de (Buneman *et al.*, 2001a)). Ainsi, nos clés sont définies de façon indépendante du type du document (ce qui n'évite pas d'éventuelles incohérences si un type est défini par ailleurs).

5.3. *Dépendances : syntaxe et sémantique*

Dans (Lee *et al.*, 2002) nous trouvons un des premiers travaux sur les *XDF*. Les idées proposées sont intéressantes mais les concepts ne sont pas précis. Par exemple, les auteurs n'ont pas défini leur notion d'égalité. Dans (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a, Liu *et al.*, 2003, Hartmann *et al.*, 2006, Wang *et al.*, 2005) nous trouvons des propositions plus précises pour les *XDF*, qui s'appuient sur des résultats établis dans le domaine relationnel. Les points de divergence et d'accord entre plusieurs de ces travaux ont été discutés dans (Hartmann *et al.*, 2006, Wang *et al.*, 2005).

Des petites variations existent entre les langages de chemins proposés dans (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a, Wang *et al.*, 2005). Le langage de

chemins dans (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a) spécifie seulement des chemins simples. Le langage de chemins de (Wang *et al.*, 2005) a plus de pouvoir d'expression. Dans (Wang *et al.*, 2005) les chemins peuvent être simples, composés, ascendants ou descendants. Un chemin simple est descendant et ne contient pas de *wildcards* alors qu'un chemin composé consiste en un chemin ascendant suivi d'un chemin descendant. Dans toutes ces approches le langage de chemins utilisé correspond à un langage de requête unaire. L'approche de (Hartmann *et al.*, 2006) est différente car le langage de chemins utilisé correspond à des requêtes n-aires. En effet, la requête de chemin est représentée par un arbre.

Dans tous les travaux cités auparavant ((Arenas *et al.*, 2002b, Buneman *et al.*, 2001a, Hartmann *et al.*, 2006, Vincent *et al.*, 2004a, Wang *et al.*, 2005)) les chemins définis dans un langage donné sont ensuite instanciés sur le document XML. Pour cela chaque auteur propose une définition compatible avec ses langages de chemin. Par exemple, la définition de *XDF* de (Arenas *et al.*, 2002b) introduit la notion de *tree tuple* qui est une fonction qui associe à un chemin une position d'un arbre XML. À partir de cette notion, les auteurs définissent une arborescence sur l'arbre XML et considèrent des n-uplets (similaires à nos n-uplets de la définition 3.2) qui sont comparés pour vérifier une *XDF*.

Le concept de motif correspond à différentes formalisations proposées pour décrire la sémantique des contraintes d'intégrité. Ainsi dans (Arenas *et al.*, 2002b), les auteurs utilisent l'ensemble de tous les chemins dans une DTD *D*, noté *Paths(D)*, dans (Hartmann *et al.*, 2006) il s'agit de *graphe-schéma* et dans (Liu *et al.*, 2003, Vincent *et al.*, 2004a) les auteurs définissent un ensemble de *chemins légaux*. La notion d'instance de motif correspond également à celle de projection utilisée dans (Benzaken *et al.*, 2006).

Dans (Vincent *et al.*, 2004a) le concept d'un ensemble de chemins simples et fermé sur préfixe correspond au concept de motif que nous avons utilisé (définition 2.6). La définition de *XDF* dans (Hartmann *et al.*, 2006) est faite sur les concepts d'homomorphisme, d'isomorphisme et d'arbre d'un document XML. L'isomorphisme est utilisé pour déterminer l'équivalence de deux sous-arbres et cette équivalence est similaire à l'égalité par valeur à partir de la racine de l'arbre (définition 2.11). Même si le point de démarrage (un langage de requêtes n-aires) est différent de celui de (Arenas *et al.*, 2002b) (un langage de requêtes unaires), les deux approches sont similaires car les dépendances fonctionnelles sont définies via des structures arborescentes (des parcours arborescents). Une originalité de (Wang *et al.*, 2005) est l'utilisation de différents types d'égalité.

Les documents XML incomplets sont traités dans (Arenas *et al.*, 2002b) et l'une ou l'autre des politiques décrites ci-dessus peut être appliquée. Dans (Vincent *et al.*, 2004a) la *satisfaction forte* est adoptée. Dans (Hartmann *et al.*, 2006) les valeurs manquantes sont caractérisées car l'axiomatisation pro-

posée ne s'applique qu'à des XDF portant sur des documents complets (sans *null*).

5.4. *Analyse*

La question de la satisfaction des contraintes (*étant donné un ensemble C de contraintes, existe t-il un arbre XML (fini) T qui satisfait C ?*) et la question de l'implication (*étant donné un ensemble C de contraintes et une contrainte c, est-ce que n'importe quel document (fini) qui satisfait C satisfait aussi c ?*) ont des applications importantes, que ce soit pour assurer la viabilité d'un ensemble de contraintes ou pour minimiser le nombre de contraintes à vérifier lors de l'exécution d'une validation. Dans le cadre du modèle relationnel, les travaux sur les problèmes de la satisfaction et de l'implication sont nombreux (de bons tours d'horizon se trouvent dans (Abiteboul *et al.*, 1995, Levene *et al.*, 1999)).

Dans le cadre de XML, ces problèmes sont également étudiés. Comme expliqué dans (Buneman *et al.*, 2001a, Buneman *et al.*, 2003a) certaines propositions de clé ne sont pas satisfaisables de façon finie selon le langage utilisé et les restrictions d'existence et d'unicité imposées (ces restrictions n'existent pas pour les autres contraintes). Par exemple, une clé $K = (//, \{k\})$ où le nœud contexte est la racine et le nœud cible est tout autre nœud de l'arbre, impose à tout élément d'avoir une clé k, y compris tout élément ayant le nom k. En d'autres termes, cette contrainte de clé impose une chaîne infinie de k nœuds et ainsi il n'existe pas un document fini capable de satisfaire cette contrainte. Il est possible d'éviter ces cas d'indécidabilité en restreignant les définitions de clés, par exemple en ne permettant pas de définir une cible avec // (c'est le cas avec *PL*), ou en imposant que tous les chemins clé aboutissent en des positions porteuses de valeur (attributs ou *data*). Dans (Buneman *et al.*, 2003a) se trouve par ailleurs la preuve de la décidabilité en PTIME du problème de l'implication (finie) pour les clés.

Il y a dans (Fan *et al.*, 2003) une étude des problèmes de la satisfaction et de l'implication finie des clés et des clés étrangères sur des langages de contraintes qui ne permettent la définition des clés et des clés étrangères qu'en termes d'attributs XML.

Une axiomatisation finie pour une classe de *XDF* est proposée dans (Hartmann *et al.*, 2006). Cette classe de *XDF* ne permet pas d'exprimer des contraintes concernant des nœuds internes car seule l'égalité de valeurs est utilisée, de plus les informations incomplètes ne sont pas considérées. Même si (Arenas *et al.*, 2002b, Vincent *et al.*, 2004a, Wang *et al.*, 2005, Arenas, 2006) définissent des formes normales pour XML, ces articles ne proposent pas une axiomatisation pour les *XDF*. Le problème de l'implication des *XDF* en présence de contraintes de schéma (DTD) est abordé dans (Arenas *et al.*, 2002b). On trouve dans (Vincent *et al.*, 2004c) une axiomatisation pour les *XDI*.

5.5. *SGBD relationnels*

Il existe des travaux dont l'objectif est d'arriver à exploiter la performance des SGBD relationnels pour la vérification des clés et clés étrangères, tout en travaillant avec des documents XML (Chen *et al.*, 2002a, Davidson *et al.*, 2003, Lee *et al.*, 2006, Davidson *et al.*, 2007). Cependant, il est montré dans (Davidson *et al.*, 2003, Davidson *et al.*, 2007) qu'il peut être impossible de décrire efficacement les contraintes XML dans la version relationnelle, en particulier les clés et les clés étrangères, même pour des transformations du format XML vers le format relationnel triviales. La question du stockage des documents XML dans un format relationnel est en soi complexe du fait de la multitude de solutions possibles : dans (Chen *et al.*, 2002a) il est noté que le meilleur choix ne serait pas le même selon que l'on voudrait favoriser l'efficacité de la maintenance des contraintes ou bien celle du calcul de requêtes. Dans (Krishnaprasad *et al.*, 2004) la difficulté du calcul de requêtes posées sur une structure hiérarchique lorsque celle-ci est stockée dans un format relationnel est analysée. L'implémentation des mises à jour est également difficile, car il faut s'assurer qu'il n'existe qu'une seule traduction, sans effet de bord, de toute mise à jour sur le document XML vers des mises à jour sur son stockage relationnel. Dans (Braganholo *et al.*, 2003) il est montré que pour certains documents XML trouver cette traduction reste un problème ouvert.

6. Conclusion

Les contraintes d'intégrité pour XML sont étudiées depuis les années 2000, mais dans les faits encore peu utilisées. On trouve dans (Fan, 2005) un tour d'horizon de plusieurs propositions de spécification, d'analyse et d'applications de telles contraintes. Dans le présent article, nous avons également décrit une famille importante de contraintes, les dépendances. Pour ce faire, nous avons proposé un formalisme homogène, exprimant les contraintes par des ensembles de chemins qui permettent de désigner les différentes parties du document XML concernées. De cette façon, les contraintes d'intégrité peuvent être décrites et vérifiées indépendamment d'un schéma, la structure requise étant donnée uniquement par les ensembles de chemins définissant les contraintes. Par rapport à un schéma complet, cette spécification ne contient pas la notion d'ordre entre les fils d'un nœud, ni de cardinalité (options, répétitions) d'un nœud, mais elle est suffisante pour les contraintes d'intégrité.

L'intérêt du formalisme utilisé est d'être suffisamment simple et clair pour permettre d'exprimer les différentes propositions de contraintes d'intégrité dans un même cadre, celui des *motifs* et de leurs instances. Cette notion nous permet de définir précisément l'interprétation de chaque type de contrainte, y compris en cas d'information incomplète pour ce qui concerne les dépendances fonctionnelles et les dépendances d'inclusion. En cas d'information incomplète, pour la validation d'un document, nous nous sommes plaçées

dans l'optique de la satisfaction faible, en considérant que les valeurs *null* sont différentes les unes des autres.

Nous avons décrit la syntaxe et la sémantique des clés et clés étrangères, cas particuliers des dépendances fonctionnelles et (respectivement) des dépendances d'inclusion et nous avons esquissé les principes d'une vérification automatique d'un document par rapport à des clés, qui s'appuie sur l'utilisation d'automates finis et d'une grammaire attribuée.

Un certain nombre de perspectives sont envisageables concernant le formalisme proposé. On pourrait en particulier considérer des expressions XPath comportant des prédicats ou des directions horizontales dans un chemin. Jusque-là, XML Schema utilisant un langage aussi réduit que *PL*, nous avons considéré notre syntaxe adaptée à la plupart des cas pratiques. Néanmoins, si une application nécessitait de pouvoir utiliser des prédicats dans les chemins qui spécifient une contrainte, alors le concept de motif (et d'instance de motif), devrait être adapté. Une autre perspective d'extension du langage pour la spécification des parties de documents concernées par la contrainte serait d'adopter un langage de requêtes arborescentes (n-aires) plutôt qu'un langage de chemins (requêtes unaires).

Les auteurs remercient les relecteurs anonymes qui leur ont permis d'améliorer significativement cet article.

7. Bibliographie

Abiteboul S., Buneman P., Suciu D., *Data on the Web - From Relations to Semistructured Data and XML*, Morgan Kaufmann Publishers, 2000.

Abiteboul S., Hull R., Vianu V., *Foundations of Databases*, Addison-Wesley Publishing Company, 1995.

Abiteboul S., Segoufin L., Vianu V., « Representing and Querying XML with incomplete information », *ACM Symposium on Principles of Database System*, 2001.

Abrão M. A., Bouchou B., Halfeld Ferrari M., Laurent D., Musicante M. A., « Incremental Constraint Checking for XML Documents », *XSym*, number 3186 in *LNCS*, 2004.

Aho A. V., Sethi R., Ullman J. D., *Compilers : principles, techniques, and tools*, Addison-Wesley, 1988.

Alechina N., Semi-structured information : a modal logic approach, Technical report, Tech. Report CSR-97-08, School of Computer Science, The University of Birmingham, 1997.

Arenas M., « Normalization theory for XML », *SIGMOD Record*, vol. 35, n° 4, p. 57-64, 2006.

Arenas M., Fan W., Libkin L., « On Verifying Consistency of XML Specifications », *ACM Symposium on Principles of Database System*, 2002a.

Arenas M., Libkin L., « A Normal form for XML Documents », *ACM Symposium on Principles of Database System*, 2002b.

Atzeni P., Morfuni N. M., « Functional Dependencies and Constraints on Null Values in Database Relations », *Information and Control*, 1986.

Beeri C., Bernstein P., « Computational problems related to the design of normal form relational schema », *ACM Trans. on Database Systems*, vol. 4, n° 1, p. 30-59, 1979.

Benedikt M., Bruns G., Gibson J., Kuss R., Ng A., « Automated Update Management for XML integrity constraints », *Program Language Technologies for XML (PLANX02)*, 2002.

Benedikt M., Fan W., Kuper G. M., « Structural Properties of XPath Fragments. », *ICDT*, 2003.

Benzaken V., Castagna G., Colazzo D., Nguyen K., « Type-Based XML Projection », *Proceedings of the VLDB Conference*, 2006.

Bidoit N., Cerrito S., Thion V., « A First Step towards Modeling Semistructured Data in Hybrid Multimodal Logic », *Journal of Applied Non-Classical Logics*, vol. 14, n° 4, p. 447-475, 2004.

Bidoit N., Colazzo D., « XML schemas capturing references », *BDA*, 2006.

Bidoit N., Colazzo D., « Testing XML constraint satisfiability », *Electr. Notes Theor. Comput. Sci.*, vol. 174, n° 6, p. 45-61, 2007.

Bonifati A., Ceri S., Paraboschi S., « Active rules for XML : A new paradigm for E-services. », *VLDB Journal*, vol. 10, n° 1, p. 39-47, 2001.

Bouchou B., Cheriat A., Ferrari M. H., Laurent D., Lima M. A., Musicante M. A., « Efficient Constraint Validation for Updated XML Databases », *Informatica*, vol. 31, p. 285-309, 2007.

Bouchou B., Halfeld Ferrari M., Musicante M. A., « Tree automata to verify XML key constraints », *WebDB*, 2003.

Braganholo V., Davidson S., Heuser C. A., « On the updatability of XML views over relational databases », *Web and Databases (WebDB)*, San Diego, CA, USA, June, 2003.

Bray T., Paoli J., Sperberg-McQueen C. M., Maler E., Yergeau F., « Extensible markup language (XML) 1.0 (Third Edition). W3C Recommendation », http ://www.w3.org/TR/2004/REC-xml-20040204/, 2004.

Buneman P., Davidson S. B., Fan W., Hara C. S., Tan W. C., « Reasoning about keys for XML. », *Inf. Syst.*, vol. 28, n° 8, p. 1037-1063, 2003a.

Buneman P., Davidson S., Fan W., Hara C., Tan W., « Reasoning about keys for XML », *Information Systems*, 2003b.

Buneman P., Davidson S., Fan W., Hara C., Tan W. C., « Keys for XML », *WWW10, May 2-5*, 2001a.

Buneman P., Fan W., Simeon J., Weinstein S., « Constraints for semistructured data and XML », *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 30, n° 1, p. 47-54, 2001b.

Casanova M. A., Fagin R., Papadimitriou C. H., « Inclusion dependencies and their interaction with functional dependencies », *PODS'82 : Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, p. 171-176, 1982.

Cattell R. G., *The Object Database Standard : ODMG 3.0*, Morgan Kaufmann, 2000.

Chen Y., Davidson S., Zheng Y., « Constraints Preserving Schema Mapping from XML to Relations », *WebDB*, p. 7-12, 2002a.

Chen Y., Davidson S., Zheng Y., « XKvalidator : A Constraint Validator for XML », *Proceedings of ACM Conference on Information and Knowledge Management*, 2002b.

Clark J., DeRose S., « XML Path Language (XPath) - Version 1.0 », W3C Working Draft, 1999. Available at http ://www.w3.org/TR/xpath, 1999.

Clark J., Murata M., « RELAX-NG Specification », http ://relaxng.org/spec-20011203.html, 2001.

Codd E. F., « Extending the database relational model to capture more meaning », *ACM SIGMOD International Conference on Management of Data*, 1979.

Davidson S. B., Fan W., Hara C. S., Qin J., « Propagating XML Constraints to Relations. », *ICDE*, p. 543-, 2003.

Davidson S., Fan W., Hara C., « Propagating XML Constraints to Relations », *J. Comput. Syst. Sci.*, vol. 73, n° 3, p. 316-361, 2007.

Demri S., « (Modal) Logics for Semistructured Data (invited talk) », *Third Workshop on Methods for Modalities (M4M-3)*, 2003.

Deutsch A., Tannen V., « XML queries and constraints, containment and reformulation », *Theoretical Computer Science*, 2005.

Fan W., « XML Contraints : Specification, Analysis, and Applications (invited talk) », *LAIC*, 2005.

Fan W., Libkin L., « On XML integrity constraints in the presence of DTDs », *ACM Symposium on Principles of Database Systems*, p. 114-125, 2001.

Fan W., Libkin L., « On XML integrity constraints in the presence of DTDs », *Journal of the ACM*, vol. 49, n° 3, p. 368-406, 2002.

Fan W., Simeon J., « Integrity constraints for XML », *Journal of Computer and System Science (JCSS)*, vol. 66, n° 1, p. 254-291, 2003.

Halfeld Ferrari Alves M., Aspects dynamiques de XML et spécification des interfaces de services web avec PEWS , Technical report, HDR - Laboratoire d'informatique (LI) de l'Université François-Rabelais de Tours, 2007.

Hartmann S., Trinh T., « Axiomatising Functional Dependencies for XML with Frequencies. », *Foundations of Information and Knowledge Systems (FoIKS), 4th International Symposium.*, p. 159-178, 2006.

Hopcroft J. E., Motwani R., Ullman J. D., *Introduction to Automata Theory Languages and Computation*, second edn, Addison-Wesley Publishing Company, 2001.

Imielinski T., « Incomplete Information in Logical Databases », *IEEE Data Engineering Bulletin*, 1989.

Imieliński T., Lipski W., « Incomplete Information in Relational Databases », *Journal of the ACM*, vol. 31, n° 4, p. 761-791, 1984.

Kot L., White W., « Characterization of the Interaction of XML functional dependencies with DTDs », *ICDT*, 2007.

Krishnaprasad M., Liu Z. H., Manikutty A., Warner J. W., Arora V., Kotsovolos S., « Query Rewrite for XML in Oracle XML DB. », *VLDB*, p. 1122-1133, 2004.

Lee M. L., Ling T. W., Low W. L., « Designing Functional Dependencies for XML », *Extending Database Technology*, p. 124-141, 2002.

Lee Q., Bressan S., Rahayu J. W., « XShreX : Maintaining Integrity Constraints in the Mapping of XML Schema to Relational. », *DEXA Workshops*, p. 492-496, 2006.

Levene M., Loizou G., *A guided tour of relational databases and beyond*, Springer-Verlag, 1999.

Lien Y. E., « On the Equivalence of Database Models », *Journal of the ACM*, 1982.

Lima M. A. V., Maintenance incrémentale des contraintes d'intégrité en XML, Thèse de doctorat, Université François-Rabelais de Tours, 2007.

Lipski W., « On Databases with Incomplete Information », *Journal of the ACM*, vol. 28, n° 1, p. 41-70, 1981.

Liu J., Vincent M. W., Liu C., « Functional Dependencies, from Relational to XML », *Ershov Memorial Conference*, p. 531-538, 2003.

Maier D., « Minimum covers in the relational database model », *J. ACM*, vol. 27, n° 4, p. 664-674, 1980.

Maier D., Mendelzon A., Sagiv Y., « Testing implications of data dependencies », *ACM Trans. On Database Systems*, vol. 4, n° 4, p. 455-469, 1979.

Mitchell J. C., « The implication problem for functional and inclusion dependencies », *Inf. Control*, vol. 56, n° 3, p. 154-173, 1983.

Shao F., Novak A., Shanmugasundaram J., « Triggers over XML views of relational data. », *Proceedings of the 31st VLDB Conference*, 2005.

Thion V., Bases de données, contraintes d'intégrité et logiques modales, Thèse de doctorat, Université Paris-Sud (Orsay), 2004.

Thompson H. S., Sperberg-McQueen C. M., Gao S., Mendelsohn N., Beech D., Maloney M., « XML Schema 1.1 Part 1 : Structures. W3C Working Draft 31 August 2006 », http ://www.w3.org/TR/xmlschema11-1/, 2006.

Vincent M. W., Liu J., Liu C., « Strong functional dependencies and their application to normal forms in XML », *ACM Transactions on Database Systems*, 2004a.

Vincent M. W., Schrefl M., Liu J., Liu C., Dogen S., « Generalized Inclusion Dependencies in XML », *Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference*, p. 224-233, 2004b.

Vincent M. W., Schrefl M., Liu J., Liu C., Dogen S., « Generalized Inclusion Dependencies in XML. », *Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference (APWeb)*, p. 224-233, 2004c.

Wang J., Topor R., « Removing XML data redundancies using functional and equality-generating dependencies », *Proceedings of the 16th Australasian Database Conference*, 2005.