Requêtes Arbres Régulières pour l'analyse de dépendances entre Vues et Mises à jour de documents XML

Hicham IDABAL — Françoise GIRE

Centre de Recherche en Informatique
Université Paris 1 Panthéon Sorbonne
90 rue de Tolbiac, 75634 Paris Cedex 13
{hidabal; gire}@univ-paris1.fr

RÉSUMÉ. *Dans ce papier nous étudions le problème classique de l'impact d'une mise à jour sur une vue, dans le cadre de données semi-structurées. Nous faisons les hypothèses suivantes: (i) le document source est modélisé par un arbre ordonné étiqueté par des symboles à arités variables, (ii) une vue $\mathcal{V}$ est une requête arbre dont l'évaluation sur le document source fournit la vue partielle souhaitée du document (iii) une classe de mises à jour $\mathcal{C}$ est également donnée par une requête arbre sélectionnant les nœuds à modifier. Nous étudions alors le problème suivant : étant données une requête de vue $\mathcal{V}$ et une classe de mise à jour $\mathcal{C}$, est-il possible de détecter si la vue $\mathcal{V}$ est indépendante de toute mise à jour q de $\mathcal{C}$? Nous montrons que le problème est en général PSPACE-difficile. Nous exhibons une condition suffisante évaluable en temps polynomial assurant l'indépendance d'une vue $\mathcal{V}$ par rapport à une classe de mises à jour $\mathcal{C}$. Nous montrons ensuite que le problème est polynomial pour la classe des requêtes de vues linéaires. Nous établissons également que le modèle de requête arbre choisi pour exprimer $\mathcal{V}$ et $\mathcal{C}$, bien qu'incomparable avec XPath, permet d'exprimer les requêtes de CoreXPath positif.*

ABSTRACT. *In this paper we study the classical problem of the impact of an update on a view defined over semi-structured data. We adopt the following working hypotheses: (i) the source document is modeled by an unranked, labeled, ordered tree, (ii) a view $\mathcal{V}$ is a tree query whose evaluation on the source document provides a desired partial view of the document, (iii) a class of updates $\mathcal{C}$ is also given by a tree query selecting the nodes to modify. We then study the following problem: given a view query $\mathcal{V}$ and a class of updates $\mathcal{C}$, is it possible to detect if the view $\mathcal{V}$ is independent of each update q in $\mathcal{C}$? We show that the problem is in general PSPACE-hard. We propose a sufficient condition evaluable in polynomial time ensuring the independence of a view $\mathcal{V}$ with respect to a class of updates $\mathcal{C}$. We then consider the class of linear view queries for which the problem becomes polynomial. We also show that the tree query model chosen to express $\mathcal{V}$ and $\mathcal{C}$, is incomparable with XPath but is able to capture positive queries of CoreXPath.*

MOTS-CLÉS : *Données semi-structurées, XML, Requêtes ,Vues, Mises à jour, Automates d'arbres.*

KEYWORDS: *Semi-structured Data, XML, Queries, Updates, Views, Tree Automata*

RSTI - ISI – 15/2010. Bases de données avancées, pages 113 à 137

1. Introduction

Motivés par le rôle-clé joué par XML comme standard des échanges d'information sur le web, les travaux portant sur la gestion des données semi-structurées foisonnent dans la littérature, et font de ce thème l'un des domaines les plus actifs de la recherche en informatique aujourd'hui. Si les aspects statiques de la gestion des données semi-structurées ont largement été abordés dans la littérature, les aspects dynamiques liés au caractère évolutif des données dans le temps, ont été moins étudiés. Sur le web, les données présentent cependant un caractère particulièrement variable et la prise en compte de cette variabilité dans la gestion de l'information est indispensable. Dans ce papier nous nous intéressons au problème de l'impact d'une mise à jour sur une vue définie sur des données semi-structurées. Une vue permet de présenter partiellement les informations issues d'un ou plusieurs documents sources afin de répondre à des besoins de personnalisation ou de confidentialité. Lorsque les données sources sont volumineuses, le coût de l'évaluation de la vue peut être élevé et il est donc important de ne pas recalculer la vue si cela ne s'avère pas nécessaire. Le problème de l'impact consiste précisément à détecter si, après une mise à jour des données sources, le résultat de l'évaluation de la requête de vue risque ou non d'être modifié. Dans le cas où l'on peut *a priori* obtenir une réponse négative à cette question, une nouvelle évaluation de la vue peut être évitée et par conséquent son coût économisé. Ce problème, classique dans le cadre des bases de données relationnelles [BLA 86, BLA 89, GRI 95, GUP 93, GUP 99, VIS 96] a été étudié dans le cadre des bases de données objet [ALI 03, SCH 91] et également pour des données semi-structurées [ABI 98, LIE 00, ONI 05, QUA 01, SAW 05, ZHU 98]. Cependant dans la plupart de ces travaux, le problème est étudié en supposant que l'on dispose des documents sources sur lesquels la vue est évaluée et les méthodes mises en œuvre utilisent ces documents sources.

Dans ce papier, nous adoptons un autre point de vue, selon lequel nous ne disposons pas *a priori* des documents sources mais seulement de la requête de vue $\mathcal{V}$ et de la mise à jour. Nous modélisons également la mise à jour sous la forme d'une requête $\mathcal{C}$, permettant de sélectionner les nœuds qui seront modifiés. Dans la mesure où nous ne précisons pas non plus la façon dont les nœuds sélectionnés par $\mathcal{C}$ seront modifiés, $\mathcal{C}$ représente en fait une classe de mises à jour possibles, celle des mises à jour ne modifiant que les nœuds sélectionnés par $\mathcal{C}$. L'énoncé du problème que nous étudions est alors le suivant : "Étant données une vue $\mathcal{V}$ et une classe $\mathcal{C}$ de mises à jour, déterminer si la vue $\mathcal{V}$ est indépendante de $\mathcal{C}$, c'est-à-dire si toute mise à jour q de $\mathcal{C}$ n'a aucun impact sur l'évaluation de $\mathcal{V}$, quel que soit le document source". Dans le cas où un schéma $\mathcal{S}c$ contraignant les documents sources est connu, on peut espérer que la connaissance de $\mathcal{S}c$ permette de détecter un nombre plus élevé de cas d'indépendance qu'en l'absence de schéma. Nous étudions donc également la variante suivante du problème précédent : "Étant donnés une vue $\mathcal{V}$, une classe $\mathcal{C}$ de mises à jour et un schéma $\mathcal{S}c$, déterminer si la vue $\mathcal{V}$ est indépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$, c'est-à-dire si toute mise à jour q de $\mathcal{C}$ n'a aucun impact sur l'évaluation de $\mathcal{V}$, quel que soit le document source $\mathcal{T}$, valide par rapport à $\mathcal{S}c$, sur lequel $\mathcal{V}$ est évaluée". Notre principale contribution est de fournir une condition suffisante pour qu'une vue $\mathcal{V}$ soit indépendante d'une classe de mise à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$. Cette condition est évaluable en temps polynomial par rapport à la taille des entrées $\mathcal{V}$, $\mathcal{C}$ et $\mathcal{S}c$. Nous

montrons aussi que le problème de l'indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$ est en général PSPACE-difficile. Enfin nous montrons que pour la sous-classe des requêtes de vues $\mathcal{V}$ linéaires, le problème devient polynomial (sans restriction sur le type de la requête $\mathcal{C}$ définissant la classe de mises à jour).

Dans le cas où l'indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$ n'est pas détectée par notre algorithme, d'autres techniques peuvent être utilisées pour raffiner l'analyse d'indépendance et maintenir la vue, comme par exemple les techniques de [ABI 98, ONI 05, SAW 05], qui utilisent les documents sources ou les vues matérialisées.

Par ailleurs pour modéliser aussi bien la requête de vue $\mathcal{V}$ que la classe de mises à jour $\mathcal{C}$, nous proposons dans ce travail d'utiliser le concept original de requête arbre régulière (RAR) qui est une forme d'extension des *tree pattern*s de [MIK 04]. Ce choix est motivé par l'indépendance de ce modèle formel vis-à-vis des standards, et par son expressivité.

Etat de l'art Des travaux similaires existent dans la littérature mais traitent plutôt de l'optimisation de requêtes : [BAL 04] utilisent des vues XPath matérialisées pour optimiser l'évaluation de requêtes XML et proposent un algorithme d'appariement entre ces vues matérialisées et les sous-expressions XPath d'une requête XML afin de déterminer si de telles vues peuvent être utilisées pour accélérer l'évaluation de la requête ; dans le cadre du problème général de réécriture d'une requête à partir d'une vue, [LAK 06] étudient, lorsque vue et requête sont des *tree pattern*, l'existence et la construction d'une requête '*maximale*' réécrite à partir de la vue et contenue dans la requête ; dans [ARI 07], le problème de la réécriture de requêtes à partir de vues est également étudié en présence de contraintes structurelles et de contraintes d'intégrité et en supposant que les requêtes et les vues sont exprimées dans un formalisme de *tree pattern* plus riche que celui utilisé dans [LAK 06] ; dans [SEG 05] la réécriture d'une requête à partir de vues est étudiée sous l'angle de la complétude du langage de réécriture c'est-à-dire sa capacité à permettre la réécriture de requêtes dont les évaluations sont déterminées par celles des vues. Ces travaux diffèrent du nôtre par différents aspects : (i) même s'il en est proche, le problème de la réécriture de requêtes en utilisant des vues n'est pas le même que celui de l'indépendance de requêtes de vues et de mises à jour, (ii) le langage de requête choisi est très souvent XPath, XQuery ou un formalisme de *tree pattern* tandis que nous optons ici pour un langage plus abstrait basé sur la notion de requête arbre régulière ; bien qu'incomparable avec XPath, ce langage permet cependant d'exprimer toutes les requêtes positives de CoreXPath, ainsi que des requêtes non exprimables en XPath comme "Donner tous les nœuds dont chaque ancêtre est un 'A'" ; (iii) enfin dans certaines approches, des informations provenant des documents sources sont stockées et utilisées dans les algorithmes proposés alors que notre approche n'utilise que les spécifications des requêtes de vue et de mise à jour et éventuellement un schéma structurel s'il est disponible.

Dans le contexte des mises à jour, d'autres travaux peuvent également être cités. [BEN 05] proposent un algorithme permettant de vérifier statiquement dans un programme de mises à jour, si les mises à jour peuvent être appliquées dès leur génération et sans attendre la fin de l'exécution du programme. Ce problème est très proche de notre problème d'indépendance entre requêtes car, changer l'ordre d'application des

opérations de mise à jour ne viole pas la sémantique du programme, si ces mises à jour n'interfèrent pas entre elles. L'analyse menée par [BEN 05] utilise cependant une approche très différente de la nôtre, mettant en œuvre des techniques de satisfaction d'un système particulier d'équations. Dans une approche trés similaire à la nôtre, [RAG 06] étudient, pour des documents XML, le problème de la détection de conflits entre des opérations de mise à jour spécifiées dans le langage $P^{*,//,[]}$ de *tree pattern*s introduit par [MIK 04]. Il est montré que le problème est NP-complet et un algorithme polynomial, pour la détection de conflits, est proposé dans le cas particulier des *tree pattern*s linéaires. À nouveau, notre approche diffère de celles utilisées dans tous ces travaux, principalement par le choix, comme langage de requêtes, du modèle formel des requêtes arbres régulières, qui impacte aussi bien les résultats obtenus que les techniques mises en œuvre.

La suite de l'article est organisé comme suit : la section 2 présente la notion d'indépendance entre vues et classes de mise à jour, la section 3 introduit le modèle des requêtes arbres régulières. Dans la section 4, nous analysons le problème d'indépendance entre requêtes, et nous exhibons un critère assurant cette indépendance dont la vérification et la complexité sont étudiées en section 5. Nous concluons en section 6.

2. Préliminaires

Dans ce travail les données semi-structurées (DSS) considérées sont des documents XML qui seront modélisés par des arbres ordonnés étiquetés sur un alphabet à arités variables Σ (figure 1). Dans ce modèle, les valeurs textuelles des éléments XML sont ignorées, seuls les éléments définissant la structure du document sont représentés par les nœuds de l'arbre. Formellement, un document $\mathcal{T}$ est un couple $\mathcal{T}$=(D, λ) où D est un domaine d'arbre, (*i. e.* D est un sous-ensemble de $\mathbb{N}^*$ contenant le mot vide ε et vérifiant $\forall i \in \mathbb{N}, wi \in D \Rightarrow (w \in D$ et $wj \in D, \forall j < i)$) et λ associe une étiquette $a \in \Sigma$ à chaque nœud w de D. Dans la suite, le domaine D de l'arbre $\mathcal{T}$=(D, λ) sera noté $\mathcal{N}(\mathcal{T})$. Pour chaque nœud w de $\mathcal{N}(\mathcal{T})$, nous notons $\mathcal{T}(w)$, le sous-arbre issu de w dans $\mathcal{T}$ défini par $\mathcal{T}(w) = (D_w, \lambda_w)$ avec $D_w = \{wv/v \in \mathbb{N}^*\} \cap D$ et λ_w est la restriction de λ à D_w. Pour des raisons techniques, nous supposerons d'autre part, que le nœud racine est toujours étiqueté par le symbole '/' dans chaque document $\mathcal{T}$.

2.1. *Concepts de vue et de mise à jour*

Vue : Une vue peut être définie comme une présentation partielle et réorganisée d'un ensemble de données sources, pour des besoins de confidentialité ou de personnalisation. Dans le cas de documents XML l'exécution d'une vue se compose principalement de deux étapes : la première étape consiste à extraire du (ou des) document(s) source(s) un ensemble de nœuds pertinents contenant les informations souhaitées ; la seconde étape réorganise le résultat obtenu lors de la première étape pour lui donner la structure personnalisée attendue.

Ainsi on peut modéliser une vue $\mathcal{V}$ par la composition de deux applications : $\mathcal{V} = h \circ t$, où l'application t sélectionne un ensemble de nœuds à extraire des données sources et l'application h procède à la réorganisation de ces nœuds pour obtenir le

résultat final. Cette application h ne jouant aucun rôle dans l'étude de l'indépendance d'une vue par rapport à un ensemble de mises à jour, nous assimilerons donc l'exécution d'une vue à sa première étape d'extraction de nœuds à partir des données sources c'est-à-dire $\mathcal{V} \simeq t$. Plus précisément, nous considérons dans ce travail que (1) les données sources sont constituées d'un unique document XML $\mathcal{T}$ et que (2) une requête de vue n-aire $\mathcal{V}$ exprime les conditions qui doivent être satisfaites par un n-uplet de nœuds de $\mathcal{T}$ pour être sélectionné par $\mathcal{V}$. Nous considérons que l'extraction d'un nœud w de $\mathcal{T}$ retourne le sous-arbre, noté $\mathcal{T}(w)$, issu de w dans $\mathcal{T}$. Ainsi l'exécution d'une requête de vue n-aire $\mathcal{V}$ sur le document $\mathcal{T}$ retourne un ensemble de n-uplets d'arbres de la forme $(\mathcal{T}(w_1), ..., \mathcal{T}(w_n))$ correspondant chacun à l'extraction d'un n-uplet de nœuds $(w_1, ..., w_n)$ sélectionné par $\mathcal{V}$.

Exemple 1 : Considérons le document semi-structuré $\mathcal{T}$ de la figure 1 et la requête de vue binaire $\mathcal{V}$ suivante : "Donner les (Titre,Auteur) des articles publiés dans un Journal". L'exécution de $\mathcal{V}$ sur $\mathcal{T}$ retourne : $\{(t_1, s_1), (t_1, s_2)\}$ avec $t_1 = \mathcal{T}(00110)$, $s_1 = \mathcal{T}(00111)$ et $s_2 = \mathcal{T}(00112)$, c'est-à-dire les deux couples regroupant chacun le titre de l'article correspondant au nœud 0011 avec le sous arbre issu de l'un de ses deux auteurs.

Mise à jour : Dans ce travail nous modélisons une mise à jour sur un document XML source $\mathcal{T}$ par une opération qui (1) sélectionne un ensemble de nœuds de $\mathcal{T}$ à mettre à jour et (2) remplace le sous-arbre $\mathcal{T}(w)$ issu de chaque nœud w sélectionné par un nouvel arbre. Ce type de mise à jour modélise un grand nombre de mises à jour, y compris les opérations d'insertion/suppression de sous-arbres dans le document source, si l'on considère ces dernières comme des mises à jour effectuées sur le nœud père de la position d'insertion/suppression. Ainsi une mise à jour q d'un document semi-structuré est définie par la composition, $q = f \circ \mathcal{C}$, de deux applications f et $\mathcal{C}$: l'application $\mathcal{C}$ sélectionne l'ensemble des nœuds w à mettre à jour et l'application f procède à la mise à jour, en remplaçant les sous-arbres $\mathcal{T}(w)$ issus de ces nœuds w par des arbres non vides. L'application $\mathcal{C}$ représente en fait une classe de mises à jour, celle des mises à jour qui modifient les nœuds sélectionnés par $\mathcal{C}$. Ainsi deux mises à jour q et q' font partie de la même classe de mises à jour si et seulement si elles sont définies à partir de la même application de sélection $\mathcal{C}$.

Exemple 2 : Soit les mises à jour, (q_1) "Modifier les auteurs d'articles en y ajoutant un numéro de téléphone (Tel)", et (q_2) "Modifier les auteurs d'articles en remplaçant l'élément Nom par la séquence des deux éléments (NomFamille, Prénom)". Ces deux mises à jour, q_1 et q_2, appartiennent à la même classe $\mathcal{C}$ de mises à jour qui sélectionne les auteurs d'articles pour les modifier.

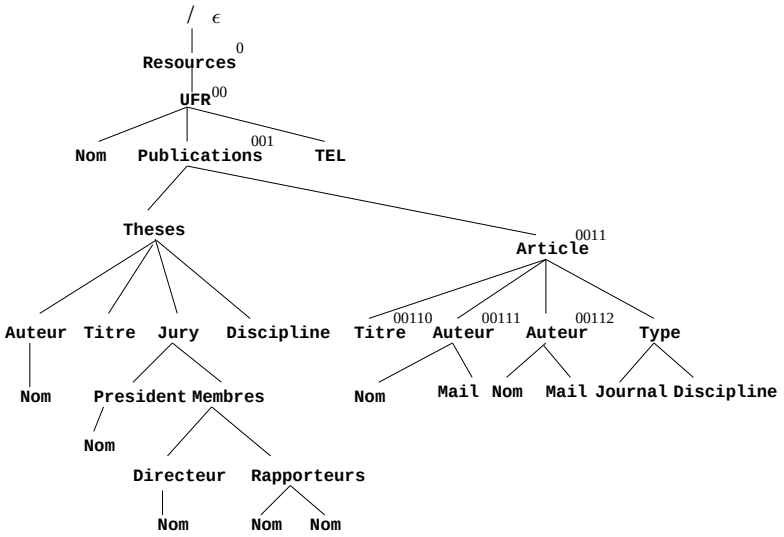


Figure 1. *Un document semi-structuré $\mathcal{T}$*

2.2. *Indépendance entre vues et classes de mises à jour*

Impact : Nous dirons qu'une mise à jour q a un impact sur une vue $\mathcal{V}$ d'un document $\mathcal{T}$ si et seulement si l'évaluation de $\mathcal{V}$ sur $\mathcal{T}$ avant et après la mise à jour q ne produit pas le même résultat, c'est-à-dire formellement, si et seulement si : $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Ainsi les requêtes de mise à jour q_1 et q_2 de l'exemple 2 ont clairement un impact sur la requête de vue $\mathcal{V}$ de l'exemple 1, puisqu'elles modifient les sous-arbres s_1 et s_2 extraits.

Indépendance : l'objectif de ce travail est de détecter directement, par l'analyse d'une requête de vue $\mathcal{V}$ et d'une mise à jour q, l'impact de cette dernière sur le résultat de l'évaluation de $\mathcal{V}$. Il est important de noter ici que l'objectif est de détecter cet impact indépendamment de la donnée du document source qui n'est pas forcément disponible lors de l'analyse. Par ailleurs, pour simplifier le problème, nous supposons que la fonction de remplacement f utilisée par la mise à jour q n'est pas donnée et par conséquent f peut être de tout type. Ainsi nous prenons l'option de nous focaliser sur la détection de l'impact éventuel, sur une vue $\mathcal{V}$, d'une classe de mises à jour $\mathcal{C}$ plutôt que d'une mise à jour q particulière. Le problème s'énonce formellement comme suit : $\mathcal{V}$ est dite indépendante par rapport à la classe de mises à jour $\mathcal{C}$ si et seulement si pour tout document source $\mathcal{T}$, pour toute mise à jour q de $\mathcal{C}$, q n'a pas d'impact sur $\mathcal{V}$ dans $\mathcal{T}$.

Notons que la notion d'indépendance utilisée dans ce contexte est différente de celle utilisée dans [LAU 01] dans le contexte des entrepôts de données où l'indépendance de mise à jour (respectivement de requête) désigne pour l'entrepôt sa capacité à se mettre à jour (respectivement à répondre à une requête) sans nécessiter d'informations supplémentaires provenant de la source. Notre notion d'indépendance est plutôt proche de la notion d'"irrelevance" utilisée dans [BLA 86, BLA 89] pour exprimer qu'une requête n'a aucun rapport avec une autre.

Exemple 3 : (impact/indépendance) Considérons sur le document $\mathcal{T}$ de la figure 1, la requête de vue $\mathcal{V}'$ suivante : "Donner les couples d'éléments (Titre, Nom) des articles publiés dans un journal". Cette requête diffère de la requête $\mathcal{V}$ de l'exemple 1 par le fait que seul le nom de l'auteur est extrait avec le titre de l'article. La mise à jour q_2 de l'exemple 2 a clairement un impact sur cette vue puisqu'elle supprime les éléments Nom. Ainsi la classe de mises à jour $\mathcal{C}$ de l'Exemple 2 a un impact sur la vue $\mathcal{V}'$ puisqu'il existe une mise à jour (q_2) de $\mathcal{C}$ qui impacte la vue $\mathcal{V}'$. Par contre, si l'on considère la classe de mise à jour $\mathcal{C}'$ qui modifie les auteurs de thèses, la vue $\mathcal{V}'$ est clairement indépendante de la classe $\mathcal{C}'$.

Indépendance dans le contexte d'un schéma : dans de nombreux cas, il est raisonnable de supposer que le schéma contraignant les données sources est connu. Cette donnée supplémentaire peut alors permettre d'affiner l'analyse d'indépendance. Notons $\texttt{valide}(\mathcal{S}c)$ l'ensemble des documents valides par rapport au schéma $\mathcal{S}c$. Dans ce contexte la définition de l'indépendance est modifiée comme suit : $\mathcal{V}$ est indépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte du schéma $\mathcal{S}c$ si et seulement si : pour tout document valide par rapport à $\mathcal{S}c$ (*i. e.* $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$) et pour toute mise à jour q de $\mathcal{C}$ conservant la validité par rapport à $\mathcal{S}c$ (*i. e.* $q(\mathcal{T}) \in \texttt{valide}(\mathcal{S}c)$), q n'a pas d'impact sur $\mathcal{V}$ dans $\mathcal{T}$.

3. Requêtes arbres régulières

Les définitions des notions de vue et de mise à jour introduites dans la section précédente font jouer un rôle central au processus de sélection de nœuds ou de n-uplets de nœuds dans un arbre. Nous prenons l'option dans ce travail d'utiliser la notion de requête arbre régulière pour modéliser ce processus. Ce choix est motivé par le caractère formel de ce modèle et donc son indépendance vis-à-vis des standards en cours. Bien qu'incomparable avec le langage $XPath$, nous montrons en section 3.3, que ce modèle permet d'exprimer toutes les requêtes exprimables par le fragment nagivationnel positif de $CoreXPath$.

3.1. *Définition*

Introduisons dans un premier temps la notion de requête arbre régulière de manière intuitive en considérant la requête binaire $\mathcal{V}$: "Donner les couples d'éléments (Titre, Auteur) de tous les articles" et son évaluation sur le document semi-structuré $\mathcal{T}$ de la figure 1. Cette requête peut être représentée par la requête arbre régulière $\mathscr{R}$ de la figure 2 qui indique de manière schématique les conditions d'extraction d'un couple de nœuds (w, w') de $\mathcal{T}$: (w, w') est extrait de $\mathcal{T}$ si et seulement s'il est possible de réaliser un plongement p de l'arbre $\mathscr{R}$ dans le document $\mathcal{T}$, de telle sorte que (voir figure 2) (a) les nœuds grisés 00 et 01 de $\mathscr{R}$ sont précisément associés par p aux nœuds w et w' c'est-à-dire $p(00) = w$, $p(01) = w'$ et , (b) pour chaque arc (w_1, w_2) de la requête $\mathscr{R}$ il existe un chemin dans le document source de $p(w_1)$ à $p(w_2)$ dont

la suite des étiquettes satisfait les contraintes exprimées par l'expression régulière étiquetant l'arc (w_1, w_2) dans $\mathscr{R}$. Formellement, une requête arbre régulière (RAR) sur l'alphabet Σ est donc un arbre dont les arcs sont étiquetés par des expressions régulières sur Σ. Nous en donnons ci-après la définition précise. Le langage régulier de mots de Σ^* défini par une expression régulière $\mathcal{E}$ est noté $L(\mathcal{E})$.

Définition 1. *Requête arbre régulière n-aire (RAR) : Soit Σ un alphabet fini d'étiquettes. Une requête arbre régulière n-aire $\mathscr{R}$ sur Σ est définie par $\mathscr{R} = (\mathscr{T}, \vec{s})$ où $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$ est appelé arbre de $\mathscr{R}$ ($N \subseteq \mathbb{N}^*$ est le domaine de l'arbre, $M \subseteq N \times N$ est l'ensemble de ses arcs, $\mathcal{E} : M \longrightarrow REG(\Sigma)$ est une application qui associe à chaque arc (w, w') de M une expression notée $\mathcal{E}_{(w,w')}$ qui est, soit vide, soit régulière propre, i. e. $L(\mathcal{E}_{(w,w')}) \subset \Sigma^+$), et $\vec{s}$=$(w_1, ..., w_n)$ est un n-uplet de nœuds spécifiques de N, représentant les n-uplets de nœuds (non nécessairement distincts) à sélectionner. Lorsque $\vec{s}$ est réduit à un seul nœud (n=1), on dit que $\mathscr{R}$ est monadique.*

La taille de $\mathscr{R}$ notée $|\mathscr{R}|$ est définie par : $|\mathscr{R}| = |N| + \Sigma_{e \in M} |\mathcal{A}_e|$ où $\mathcal{A}_e$ est un automate fini de mots quelconque associé à l'expression régulière $\mathcal{E}_e$ et $|\mathcal{A}_e|$ désigne la taille de $\mathcal{A}_e$. Pour détailler le contenu de l'arbre $\mathscr{T}$, on adopte l'écriture suivante :

$$\begin{aligned} \mathscr{T} \quad = \quad & (\epsilon = [\mathcal{E}_{(\epsilon,0)} \rightarrow 0]; \\ & 0 = [\mathcal{E}_{(0,00)} \rightarrow 00, ..., \mathcal{E}_{(0,0k_0)} \rightarrow 0k_0]; \\ & ... \\ & w = [\mathcal{E}_{(w,w0)} \rightarrow w0, ..., \mathcal{E}_{(w,wk_w)} \rightarrow wk_w]; ...) \end{aligned}$$

Cette écriture donne pour chaque nœud w de N, la liste des expressions régulières $\mathcal{E}_{(w,wj)}$, $0 \leqslant j \leqslant k_w$ associées à l'ensemble, noté $Out(w)$, des arêtes sortantes du nœud w, (k_w+1) est l'arité du nœud w *i. e.* $(k_w + 1) = |Out(w)|$. Compte tenu du fait que les racines (étiquetées par /) des arbres représentant les documents XML ont un unique fils, nous ne considérons dans ce travail que les requêtes arbres régulières $\mathscr{R}$ pour lesquelles $Out(\epsilon)$ est réduit à un seul nœud *i. e.* $k_\epsilon = 0$.

La figure 3 donne deux exemples de requêtes RAR. Dans cette figure, les nœuds de sélection de $\vec{s}$ sont grisés. La sémantique de ces deux requêtes est donnée plus loin en section 3.2.

3.2. *Évaluation d'une requête arbre régulière*

L'évaluation d'une requête RAR n-aire sur un document semi-structuré est basée sur la notion de plongement.

Définition 2. *Un plongement, dans un document semi-structuré $\mathcal{T}$, d'une requête RAR $\mathscr{R} = (\mathscr{T}, \vec{s})$ où $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$, est une fonction p injective totale de N dans $\mathcal{N}(\mathcal{T})$ vérifiant :*

– l'image du nœud racine ϵ de l'arbre $\mathscr{T}$ est le nœud racine de $\mathcal{T}$ étiqueté par '/'.

– $\forall w, w' \in N$, si $w < w'$ alors $p(w) < p(w')$.

– $\forall e = (w, w') \in M$, il existe dans $\mathcal{T}$ un chemin P_e de $p(w)$ à $p(w')$ tel que :

(a) la suite, notée $\sigma(e)$, des étiquettes des nœuds de P_e, excluant celle de $p(w)$ et

incluant celle de $p(w')$, est un mot du langage régulier $L(\mathcal{E}_{(w,w')})$ et,
(b) si, dans $\mathcal{T}$, $e_1 = (w, wi)$ et $e_2 = (w, wj)$ sont deux arêtes distinctes de $Out(w)$ alors les chemins P_{e_1} et P_{e_2} n'ont pas de préfixe commun.

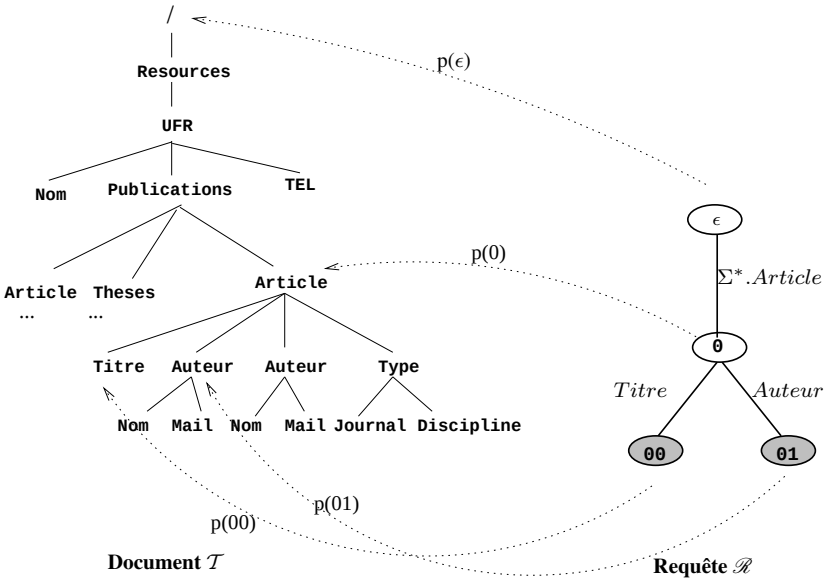
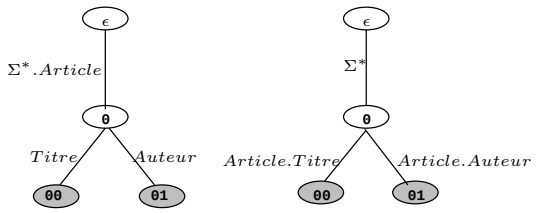


Figure 2. *Plongement de la requête $\mathscr{R}$ dans le document $\mathcal{T}$*

La condition précédente (b) de la définition 2 est illustrée par les requêtes RAR $\mathscr{R}$ et $\mathscr{R}'$ de la figure 3. Bien que $\mathscr{R}$ et $\mathscr{R}'$ aient des structures très similaires, leurs plongements, dans un document semi-structuré, diffèrent : les plongements de $\mathscr{R}$ extraient les couples d'éléments (Titre, Auteur) qui sont fils d'un même nœud Article, alors que ceux de $\mathscr{R}'$ extraient les couples d'éléments (Titre, Auteur) qui sont fils de deux nœuds Article distincts.

Trace d'une requête RAR selon un plongement : On appelle trace de $\mathscr{R}$ dans le



(a) Requête arbre $\mathscr{R}$ **(b) Reqête arbre $\mathscr{R}'$**

Figure 3. *Requêtes arbres régulières*

document $\mathcal{T}$ selon le plongement p, le plus petit sous-arbre de $\mathcal{T}$ contenant l'image $p(N)$ et on le note $\texttt{trace}_p(\mathscr{R}, \mathcal{T})$.

Évaluation d'une requête RAR : Soit $\mathscr{R}$ une requête RAR, $\mathcal{T}$ un document semi-structuré et $\mathcal{P}$ l'ensemble de tous les plongements de $\mathscr{R}$ dans $\mathcal{T}$.

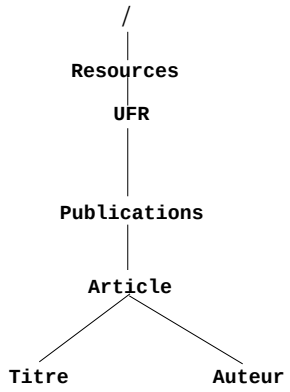


Figure 4. *Trace de la requête RAR $\mathscr{R}$ dans $\mathcal{T}$ selon p*

– Le résultat, noté $\mathscr{R}_p(\mathcal{T})$, de l'évaluation de $\mathscr{R}$ sur $\mathcal{T}$ suivant le plongement p de $\mathcal{P}$, est le n-uplet d'arbres $\mathscr{R}_p(\mathcal{T}) = (\mathcal{T}(p(w_1)), \ldots, \mathcal{T}(p(w_n)))$ où $\vec{s} = (w_1, ..., w_n)$ est le n-uplet de nœuds de sélection de $\mathscr{R}$.

– Le résultat, noté $\mathscr{R}(\mathcal{T})$, de l'évaluation de $\mathscr{R}$ sur $\mathcal{T}$ est $\mathscr{R}(\mathcal{T}) = \bigcup_{p\in\mathcal{P}} \mathscr{R}_p(\mathcal{T})$.

Projection d'une requête RAR : Soit J=$(i_1, i_2, ..., i_k)$ un sous n-uplet de (1,2,..,n). La projection suivant J d'une requête n-aire $\mathscr{R}$ est la requête k-aire, notée $\Pi_J(\mathscr{R})$, définie par : $\forall\mathcal{T}, Eval_{\mathcal{T}}(\Pi_J(\mathscr{R})) = \{\Pi_J(\vec{u})/\vec{u} \in Eval_{\mathcal{T}}(\mathscr{R})\}$.

On vérifie facilement que la projection $\Pi_J(\mathscr{R})$ d'une requête RAR $\mathscr{R} = (\mathscr{T}, \vec{s})$ est la requête RAR $\Pi_J(\mathscr{R}) = (\mathscr{T}, \Pi_J(\vec{s}))$.

3.3. *Requêtes arbres régulières et $XPath$*

La notion de requête RAR, que nous avons choisie pour modéliser le processus de sélection de nœuds ou de n-uplets de nœuds dans un arbre, est un modèle formel puissant qui permet d'exprimer certaines requêtes que le langage standard $XPath$ ne permet pas d'exprimer. Ainsi, la requête "Donner tous les nœuds dont chaque ancêtre est étiqueté par un 'a'" n'est pas exprimable en $XPath$ alors qu'elle est exprimée par la requête RAR ($\mathscr{T}$,{0}) où $\mathscr{T}$ est l'arbre ($\epsilon = [a^* \rightarrow 0]$;).

$XPath$ est cependant un standard capable de gérer à la fois la structure d'un document XML mais aussi son contenu informationnel (nœuds texte, attributs) et d'autres aspects syntaxiques du standard XML (commmentaires, instructions de traitements, etc.). Ainsi $XPath$ et le modèle des requêtes RAR sont incomparables mais nous montrons dans la suite de cette section que le fragment $CoreXPath^+$ de $XPath$ est capturé par les unions de requêtes RAR monadiques.

$CoreXPath^+$ est le fragment navigationnel de $XPath$ n'utilisant ni négation ni disjonction. Formellement, un chemin absolu P_a de $CoreXPath^+$ est défini par la grammaire :

$$\begin{aligned} P_a &\equiv /P \\ P &\equiv axis :: \lambda \parallel P/axis :: \lambda \parallel P[F] \parallel P|P \\ F &\equiv F_{un} \parallel F \; and \; F \\ F_{un} &\equiv axis :: \lambda \parallel axis :: \lambda[F] \parallel /F_{un} \end{aligned}$$

où λ appartient à $\Sigma \cup \{*\}$ et $axis$ est l'un des axes navigationnels de $XPath$: *self, child, parent, descendant, descendant_or_self, ancestor, ancestor_or_self, following_sibling, following, preceding_sibling ou preceding*.

Toute expression engendrée par le non terminal P (respectivement F, F_{un}) est appelée un *chemin relatif* (respectivement un *filtre*, un *filtre unaire*). Ainsi, tout filtre f de $CoreXPath^+$ s'écrit sous la forme (f_1 and f_2 and ... and f_k) où les f_i sont des filtres unaires (par abus de notation, on choisit k=1 si f est unaire).

Pour montrer que tout chemin de $CoreXPath^+$ est exprimable par une union finie de requêtes RAR monadiques, nous utilisons deux opérations : l'extension et le filtrage de requêtes RAR par un filtre de $CoreXPath^+$.

Extension et filtrage de requêtes RAR par un filtre de $CoreXPath^+$

Pour chaque axe navigationnel $axis$ de $XPath$ et chaque nœud u d'un document $\mathcal{T}$, on note $axis(u)$ l'ensemble des nœuds v de $\mathcal{N}(\mathcal{T})$ qui sont accessibles à partir de u en utilisant l'axe $axis$. On associe alors, à chaque filtre f unaire de $CoreXPath^+$, la relation Rel^f définie comme suit sur les nœuds d'un document $\mathcal{T}$: $\forall u, v \in \mathcal{N}(\mathcal{T})$,
- si $f = axis :: \lambda$, $Rel^f(u, v)$ est vrai si et seulement si $v \in axis(u)$ et v est étiqueté par λ
- si $f = axis :: \lambda[f_1 \; and \; ... \; and \; f_k]$ avec $k \geq 1$ et $\forall i, \; 1 \leq i \leq k, \;\; f_i$ élémentaire, $Rel^f(u, v)$ est vrai si et seulement si $Rel^{axis::\lambda}(u, v)$ est vrai et $\forall i, 1 \leq i \leq k$, il existe $w_i \in \mathcal{N}(\mathcal{T})$ tel que $Rel^{f_i}(v, w_i)$ est vrai
- si $f = /g$ avec g un filtre de $CoreXPath^+$, $Rel^f(u, v)$ est vrai si et seulement si $Rel^g(\epsilon, v)$ est vrai.

Les relations associées aux filtres unaires de $CoreXPath^+$ interviennent dans la définition des opérations d'extension et de filtrage de requêtes RAR.

Définition 3. *Soit $\mathscr{R}$ une requête RAR n-aire et f = f_1* and ... and *f_k un filtre de $CoreXPath^+$ avec $\forall i, 1 \leq i \leq k$, f_i unaire. L'extension $\mathscr{R}.f$ et le filtrage $\mathscr{R}[f]$ de $\mathscr{R}$ par f, sont des requêtes évaluées sur tout document $\mathcal{T}$ comme suit :*

-$Eval_{\mathcal{T}}(\mathscr{R}.f) = \{(u_1, ..., u_n, u_{n+1}, ..., u_{n+k}) \in \mathcal{N}(\mathcal{T})^{n+k} / (u_1, ..., u_n) \in Eval_{\mathcal{T}}(\mathscr{R})$ et $\forall j, 1 \leq j \leq k$, $Rel^{f_j}(u_n, u_{n+j})$ }
-$Eval_{\mathcal{T}}(\mathscr{R}[f]) = \{(u_1, ..., u_n) \in Eval_{\mathcal{T}}(\mathscr{R}) / \exists (u_{n+1}, ..., u_{n+k}) \in \mathcal{N}(\mathcal{T})^{n+k}$ tels que $\forall j, 1 \leq j \leq k$, $Rel^{f_j}(u_n, u_{n+j})$ }

La stabilité des unions de requêtes RAR par les opérations d'extension et de filtrage est établie par le Lemme 1. Cette propriété est cruciale pour montrer que tout chemin de $CoreXPath^+$ est exprimable par une union de requêtes RAR monadiques.

Lemme 1. *Soit f un filtre de $CoreXPath^+$ et $\mathscr{R} = (\mathscr{T}, \vec{s})$ une requête RAR, les requêtes $\mathscr{R}.f$ et $\mathscr{R}[f]$ sont exprimables par des unions de requêtes RAR.*

La preuve de ce lemme est assez technique et se fait par induction sur la longueur du filtre f. Nous en donnons les principales lignes en appendice pour le lecteur intéressé.

Proposition 1. *Tout chemin de $CoreXPath^+$ est exprimable par une union de requêtes RAR monadiques.*

Démonstration. Soit $p_a = /p$ où p est un chemin relatif de $CoreXPath^+$, on procède par induction sur la structure de p.
Cas de base ($p = axis :: \lambda$) :
On vérifie facilement que les seuls cas où l'évaluation de p_a n'est pas soit vide soit réduite à la racine '/' du document, sont ceux pour lesquels $axis \in \{$child, descendant$\}$.
- Si $p = child :: \lambda$ alors p_a est exprimable par la requête RAR monadique $\mathscr{R}_p = \{(\epsilon = [\lambda \rightarrow 0]), \{0\}\}$ qui sélectionne la racine si elle est étiquetée par λ.
- Si $p = descendant :: \lambda$ alors p_a est exprimable par la requête RAR monadique $\mathscr{R}_p = \{(\epsilon = [\Sigma^*\lambda \rightarrow 0]), \{0\}\}$ qui sélectionne les nœuds étiquetés par λ.
- Si l'évaluation de p_a ne sélectionne aucun nœud (respectivement est réduite à la racine), p_a est exprimable par la requête RAR vide $\mathscr{R}_\emptyset = \{(\epsilon = [\emptyset \rightarrow 0]), \{0\}\}$ (respectivement $\mathscr{R}_/ = \{(\epsilon), \{\epsilon\}\}$).
Etape d'induction :
- Si $p_a = /p/axis :: \lambda$ ou $p_a = /p[f]$ avec $/p$ équivalent à une union $\cup_k \mathscr{R}_k$ de requêtes RAR monadiques, on a :
$/p/axis :: \lambda$ est équivalente à $\cup_k(\mathscr{R}_k/axis :: \lambda)$ et donc à $\cup_k \Pi_2(\mathscr{R}_k.(axis :: \lambda))$.
$/p[f]$ est équivalente à $\cup_k \mathscr{R}_k[f]$. Le Lemme 1 donne donc le résultat.
- Si $p_a = /(p_1|p_2)$ avec $/p_1$ et $/p_2$ équivalentes à des unions de requêtes RAR monadiques alors le résultat est immédiat car p_a est équivalent à $/p_1 \cup /p_2$.

4. Problème d'indépendance

Dans les deux prochaines sections, nous étudions le problème de l'indépendance d'une vue $\mathcal{V}$, modélisée par une requête RAR $\mathcal{V} = (\mathscr{T}_\mathcal{V}, \vec{s_\mathcal{V}})$, par rapport à une classe de mises à jour $\mathcal{C}$ modélisée par une requête RAR $\mathcal{C} = (\mathscr{T}_\mathcal{C}, \vec{s_\mathcal{C}})$. Dans cette section, nous montrons que le problème est en général PSPACE-difficile puis, sous l'hypothèse que $\mathcal{C}$ est monadique et que le nœud de mise à jour $s_\mathcal{C}$ est une feuille de $\mathscr{T}_\mathcal{C}$, nous exhibons une condition suffisante assurant l'indépendance de $\mathcal{V}$ par rapport à la classe $\mathcal{C}$. Nous montrons ensuite en section 5 que cette condition peut être évaluée en temps polynomial.

4.1. *Un problème PSPACE-difficile*

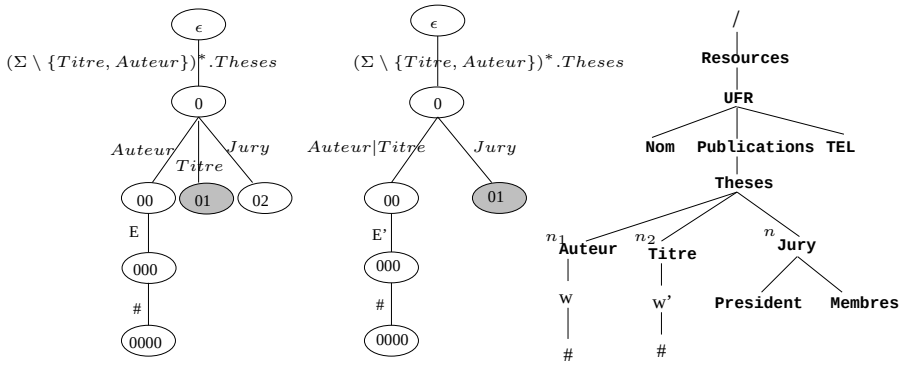
Proposition 2. *Décider si une vue $\mathcal{V} = (\mathscr{T}_\mathcal{V}, \vec{s_\mathcal{V}})$ est indépendante par rapport à une classe de mise à jour $\mathcal{C} = (\mathscr{T}_\mathcal{C}, \vec{s_\mathcal{C}})$ est un problème PSPACE-difficile.*

Démonstration. Nous réduisons le problème de l'inclusion d'expressions régulières, qui est PSPACE-difficile [MAR 04], au problème de l'indépendance d'une vue par rapport à une classe de mises à jour. Considérons deux expressions régulières E et E' sur Σ. Nous définissons deux requêtes RAR monadiques $\mathcal{V}$ et $\mathcal{C}$ comme indiqué sur la figure 5 ((a) et (b)) où '#' est une nouvelle étiquette qui n'apparaît ni dans E ni dans E', et nous montrons que $\mathcal{V}$ est dépendante de $\mathcal{C}$ si et seulement si $L(E) \nsubseteq L(E')$. Supposons que $\mathcal{V}$ soit dépendante de $\mathcal{C}$. Il existe un document $\mathcal{T}$ et une mise à jour q de $\mathcal{C}$ tels que $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Deux cas sont donc possibles : soit (a) un nœud n de $\mathcal{T}$ étiqueté par 'Jury' est extrait par $\mathcal{V}$ ($n \in \mathcal{V}(\mathcal{T})$) mais ne l'est plus après la mise à jour ($n \notin \mathcal{V}(q(\mathcal{T}))$), soit (b) un nouveau nœud n étiqueté par 'Jury' est extrait par $\mathcal{V}$ après la mise à jour ($n \in \mathcal{V}(q(\mathcal{T}))$) alors qu'il ne l'était pas avant ($n \notin \mathcal{V}(\mathcal{T})$). Dans les deux cas, on déduit que le nœud n appartient nécessairement à une trace de la requête $\mathcal{C}$ dans $\mathcal{T}$ selon un plongement p. Par conséquent n a deux nœuds frères n_1 et n_2 étiquetés respectivement par 'Auteur' et 'Titre', qui sont les images par p des nœuds 00 et 01 de $\mathcal{C}$. Si $L(E) \subseteq L(E')$, le sous-arbre issu de n_1 remplit alors les conditions imposées par la requête $\mathcal{V}$ pour l'extraction de n, dans $q(\mathcal{T})$ après la mise à jour pour le cas (a), et dans $\mathcal{T}$ avant la mise à jour pour le cas (b). Ceci contredit les faits $n \notin \mathcal{V}(q(\mathcal{T}))$ (cas (a)) et $n \notin \mathcal{V}(\mathcal{T})$ (cas (b)). Inversement si $L(E) \nsubseteq L(E')$, alors il existe un mot w de $L(E)$ qui n'appartient pas à $L(E')$. Soit d'autre part w' un mot de $L(E')$. Considérons l'arbre $\mathcal{T}_0$ de la figure 5 (c) dans lequel n_1, n_2 et n représentent les nœuds étiquetés respectivement par 'Auteur', 'Titre' et 'Jury', et le mot w (resp. w') coïncide avec la suite des étiquettes rencontrées sur le chemin allant du nœud Auteur (respectivement Titre) au nœud $\#$. Le nœud n sera extrait par $\mathcal{V}$ avant toute mise à jour de $\mathcal{C}$ car w' appartient à $L(E')$. Le nœud n_2 est modifié par toute mise à jour de $\mathcal{C}$ car w appartient à $L(E)$. Considérons maintenant la mise à jour q de $\mathcal{C}$ qui supprime le chemin $w'\#$ du sous-arbre issu de n_2. Le nœud n ne sera plus extrait par $\mathcal{V}$ après la mise à jour q car w n'appartient pas à $L(E')$. Ainsi $\mathcal{V}$ est dépendante de $\mathcal{C}$.

La Proposition 2 donne une borne inférieure pour la complexité du problème de l'indépendance d'une vue par rapport à une classe de mises à jour. Nous conjecturons que le problème est PSPACE-complet : l'existence d'un arbre, témoin de l'indépendance d'une vue $\mathcal{V}$ par rapport à une classe de mis à jour $\mathcal{C}$, et dont la construction peut être faite en espace polynomial par rapport à la taille de $\mathcal{V}$ et de $\mathcal{C}$, reste à montrer formellement.

4.2. *Un critère d'indépendance*

A partir de maintenant et jusqu'à la fin de ce travail, nous faisons l'hypothèse supplémentaire que $\mathcal{C} = (\mathscr{T}_\mathcal{C}, \vec{s_\mathcal{C}})$ est monadique et que le nœud de mise à jour $s_\mathcal{C}$ est une feuille de $\mathscr{T}_\mathcal{C}$. Cette hypothèse va en effet nous permettre d'exhiber un critère d'indépendance d'une vue $\mathcal{V}$ par rapport à une classe de mises à jour $\mathcal{C}$, évaluable en temps polynomial.



(a) Classe de mise à jour $\mathcal{C}$ (b) Vue $\mathcal{V}$ (c) $Arbre\ \mathcal{T}_0$

Figure 5. *Schéma de réduction*

Selon les définitions de la section précédente, une requête de vue $\mathcal{V}$ est dépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte du schéma $\mathcal{S}c$ si et seulement s'il existe $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$, il existe $q \in \mathcal{C}$ vérifiant $q(\mathcal{T}) \in \texttt{valide}(\mathcal{S}c)$ et il existe un n-uplet $\overrightarrow{t}$ de sous-arbres de $\mathcal{T}$ vérifiant l'une des deux assertions :
(1) $\overrightarrow{t} \in \mathcal{V}(\mathcal{T})\ et\ \overrightarrow{t} \notin \mathcal{V}(q(\mathcal{T}))$
ou (2) $\overrightarrow{t} \notin \mathcal{V}(\mathcal{T})\ et\ \overrightarrow{t} \in \mathcal{V}(q(\mathcal{T}))$.

L'assertion (1) signifie qu'un n-uplet $\overrightarrow{t}$, extrait avant la mise à jour, ne l'est plus après : il existe donc une trace $\texttt{trace}_p(\mathcal{V}, \mathcal{T})$ de $\mathcal{V}$ dans $\mathcal{T}$ selon un plongement p, et un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ permettant la mise à jour q, vérifiant l'une des conditions suivantes :

– Un nœud de $\texttt{trace}_p(\mathcal{V}, \mathcal{T})$ a été modifié par q et les conditions d'extraction de $\overrightarrow{t}$ ne sont plus satisfaites (Cas (i) figure 6). Ainsi le nœud mis à jour, $p'(s_{\mathcal{C}})$, appartient à $\mathcal{N}(\texttt{trace}_p(\mathcal{V}, \mathcal{T}))$

– Un sous-arbre de $\overrightarrow{t}$ a été modifié par q et $\overrightarrow{t}$ n'apparaît plus dans le résultat de $\mathcal{V}$ (Cas (ii) figure 6). Ainsi le nœud mis à jour, $p'(s_{\mathcal{C}})$, appartient à $\mathcal{N}(\mathcal{V}_p(\mathcal{T}))$

L'assertion (2) signifie qu'un nouveau n-uplet $\overrightarrow{t}$ est extrait après la mise à jour. Ainsi la mise à jour par q du nœud $p'(s_{\mathcal{C}})$ de $\mathcal{T}$ a créé une trace $\texttt{trace}_p(\mathcal{V}, q(\mathcal{T}))$ de $\mathcal{V}$ dans $q(\mathcal{T})$ selon un plongement p, permettant l'extraction de $\overrightarrow{t}$ (Cas (iii) figure 6). Dans ce cas, $p'(s_{\mathcal{C}})$ appartient à $\mathcal{N}(\texttt{trace}_p(\mathcal{V}, q(\mathcal{T})))$

Les assertions (1) et (2) impliquent chacune l'existence d'au moins un arbre ($\mathcal{T}$ pour l'assertion (1) et q($\mathcal{T}$) pour l'assertion (2)) satisfaisant certaines propriétés. Nous définissons formellement dans la suite le language $\mathcal{L}$ des arbres satisfaisant ces propriétés, et nous montrons (Proposition 3) que la vacuité de $\mathcal{L}$ est une condition suffisante pour l'indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$ dans le contexte du schéma $\mathcal{S}c$.

Si $\vec{t} = (\mathcal{T}_1, ..., \mathcal{T}_n)$ est un n-uplet de sous-arbres, on note $\mathcal{N}(\vec{t})$ l'union des nœuds de ces sous-arbres : $\mathcal{N}(\vec{t}) = \cup_{i=1}^{n} \mathcal{N}(\mathcal{T}_i)$.

Définition 4. *On note $\mathcal{L}$ l'ensemble des arbres $\mathcal{T}$ vérifiant :*
(i) $\mathcal{T} \in \mathtt{valide}(\mathcal{S}c)$
(ii) il existe une trace $\mathtt{trace}_p(\mathcal{V}, \mathcal{T})$ de $\mathcal{V}$ dans $\mathcal{T}$ selon un plongement p et il existe une trace $\mathtt{trace}_{p'}(\mathcal{C}, \mathcal{T})$ de $\mathcal{C}$ dans $\mathcal{T}$ selon un plongement p' telles que :
$p'(s_{\mathcal{C}}) \in \mathcal{N}(\mathtt{trace}_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$

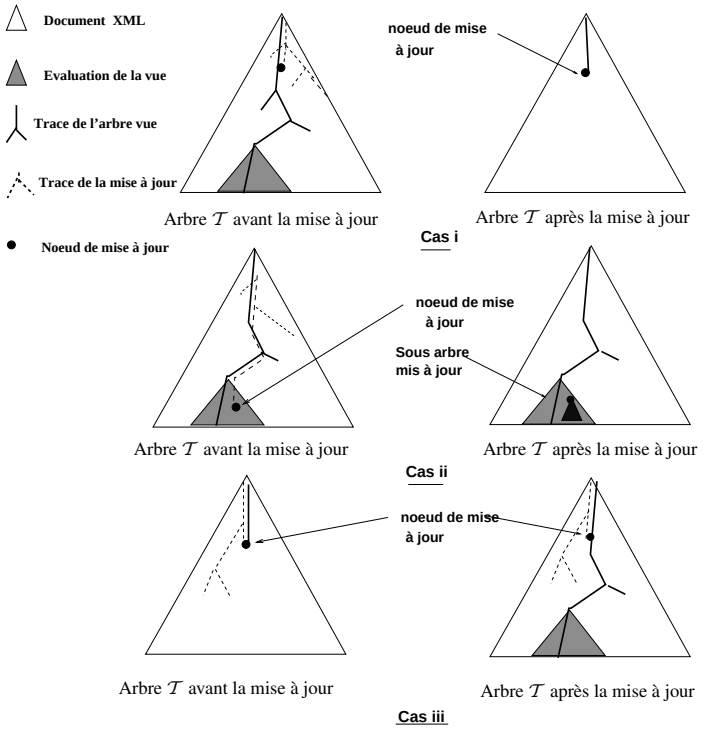


Figure 6. *Analyse de l'impact d'une mise à jour sur une vue*

Proposition 3. *Si $\mathcal{L} = \varnothing$ alors $\mathcal{V}$ est indépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$.*

Démonstration. Supposons que $\mathcal{V}$ soit dépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$. L'assertion (1) ou (2) est donc satisfaite.

Si l'assertion (1) ($\vec{t} \in \mathcal{V}(\mathcal{T})$ et $\vec{t} \notin \mathcal{V}(q(\mathcal{T}))$) est satisfaite, nous avons vu qu'il existe un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ et un plongement p de $\mathcal{V}$ dans $\mathcal{T}$ vérifiant : $p'(s_{\mathcal{C}}) \in \mathcal{N}(\mathtt{trace}_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$. Le langage $\mathcal{L}$ est donc non vide.

Si l'assertion (2) ($\vec{t} \notin \mathcal{V}(\mathcal{T})$ et $\vec{t} \in \mathcal{V}(q(\mathcal{T}))$) est satisfaite, nous avons vu qu'il existe $\mathcal{T} \in \mathtt{valide}(\mathcal{S}c)$, une mise à jour q dans $\mathcal{C}$ qui vérifie $q(\mathcal{T}) \in \mathtt{valide}(\mathcal{S}c)$, un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ permettant la mise à jour q, un plongement p de $\mathcal{V}$ dans $q(\mathcal{T})$ permettant l'extraction de $\vec{t}$ aprés la mise à jour q et $p'(s_{\mathcal{C}})$ appartient à $\mathcal{N}(\mathtt{trace}_p(\mathcal{V}, q(\mathcal{T})))$. Puisque le nœud de mise à jour $s_{\mathcal{C}}$ est une feuille de l'arbre $\mathscr{T}_{\mathcal{C}}$, le plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ subsiste dans $q(\mathcal{T})$: formellement, il existe un

plongement p'' de $\mathcal{C}$ dans $q(\mathcal{T})$ identique au plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ et vérifiant donc $p'(s_{\mathcal{C}}) = p''(s_{\mathcal{C}})$. Ainsi $p''(s_{\mathcal{C}}) \in \mathcal{N}(\mathtt{trace}_p(\mathcal{V}, q(\mathcal{T})))$ et $q(\mathcal{T})$ satisfait les propriétés des arbres de $\mathcal{L}$. Le langage $\mathcal{L}$ est donc non vide.

Remarque Le test de vacuité du langage $\mathcal{L}$ n'est pas une condition nécessaire pour que la requête de vue $\mathcal{V}$ soit indépendante de la classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$. Reprenons les requêtes $\mathcal{V}$ et $\mathcal{C}$ de la figure 5 ainsi que l'arbre $\mathcal{T}_0$, et supprimons, dans $\mathcal{V}$ et $\mathcal{C}$, les sous-arbres issus du nœud 00 et, dans $\mathcal{T}_0$, les sous-arbres issus des nœuds n_1 et n_2 : l'arbre $\mathcal{T}_0$ appartient à $\mathcal{L}$ donc $\mathcal{L}$ est non vide mais on peut facilement vérifier l'indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$.

4.3. *Requêtes arbres de vue linéaires*

Dans le cas particulier où la requête de vue est définie par un arbre linéaire, le critère énoncé en Proposition 3 devient une condition nécessaire et suffisante d'indépendance de la vue par rapport à la classe de mises à jour.

Définition 5. *Une requête RAR $\mathscr{R} = (\mathscr{T}, \vec{s})$ est dite linéaire si et seulement si l'arbre $\mathscr{T}$ est linéaire.*

Proposition 4. *Soit $\mathcal{V} = (\mathscr{T}_{\mathcal{V}}, \vec{s_{\mathcal{V}}})$ une requête de vue linéaire et $\mathcal{C} = (\mathscr{T}_{\mathcal{C}}, s_{\mathcal{C}})$ une classe de mises à jour. $\mathcal{V}$ est indépendante de la classe $\mathcal{C}$ si et seulement si $\mathcal{L}$ est vide.*

Démonstration. D'après la Proposition 3, il suffit de montrer que la vacuité du langage $\mathcal{L}$ est une condition nécessaire d'indépendance. Supposons donc $\mathcal{L} \neq \emptyset$ et soit $\mathcal{T}$ un arbre de $\mathcal{L}$. Nous montrons que $\mathcal{T}$ est un arbre témoin de la dépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$. Puisque $\mathcal{T} \in \mathcal{L}$, il existe dans $\mathcal{T}$ une trace $\mathtt{trace}_p(\mathcal{V}, \mathcal{T})$ de $\mathcal{V}$ selon un plongement p, une trace $\mathtt{trace}_{p'}(\mathcal{C}, \mathcal{T})$ de $\mathcal{C}$ selon un plongement p', et le nœud mis à jour $n = p'(s_{\mathcal{C}})$ appartient soit à $\mathcal{N}(\mathtt{trace}_p(\mathcal{V}, \mathcal{T}))$ (cas (a) de la figure 7) soit à $\mathcal{N}(\mathcal{V}_p(\mathcal{T}))$ (cas (b) de la figure 7). Le fait que la requête de vue $\mathcal{V}$ soit linéaire assure une relation d'ascendance ou de descendance entre le nœud n mis à jour et un nœud m sélectionné par $\mathcal{V}$. Dans les deux cas (a) et (b), il est alors possible d'exhiber une mise à jour q de $\mathcal{C}$ ajoutant par exemple une nouvelle feuille au sous-arbre $\mathcal{T}(n)$ de telle sorte que le sous-arbre $\mathcal{T}(m)$ soit également modifié. Le résultat de la vue est donc modifié par q. On en déduit la dépendance de la vue $\mathcal{V}$ par rapport à la classe de mises à jour $\mathcal{C}$.

5. Vérification du critère d'indépendance

Dans cette section, nous montrons que la vérification du critère d'indépendance, $\mathcal{L} = \emptyset$, est décidable en temps polynomial par rapport aux tailles de $\mathcal{V}$, $\mathcal{C}$ et $\mathcal{S}c$. Pour cela, nous montrons que $\mathcal{L}$ est un langage régulier d'arbres reconnaissable par un automate $\mathcal{A}$ dont la taille est polynomiale en les tailles de $\mathcal{V}$, $\mathcal{C}$ et $\mathcal{S}c$. Le résultat découle alors du fait que le test de vacuité d'un langage régulier d'arbres est décidable en temps polynomial en la taille d'un automate reconnaîssant ce langage. Nous commençons en section 5.1 par construire un automate reconnaissant l'ensemble des arbres contenant une trace d'une requête RAR $\mathscr{R}$, nous donnons ensuite en section 5.2, la construction de l'automate $\mathcal{A}$, et terminons enfin en étudiant la complexité des constructions précédentes.

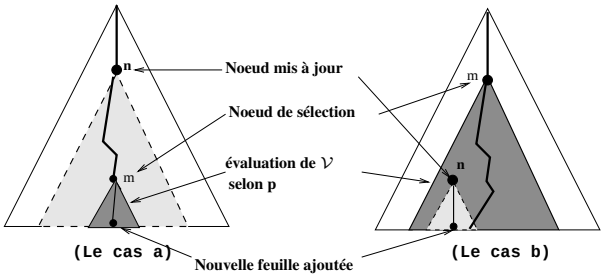


Figure 7. *Cas des vues linéaires*

5.1. *Automate reconnaissant une trace*

Soit la requête arbre régulière $\mathscr{R} = (\mathscr{T}, \vec{s})$ avec $\mathscr{T} = (\Sigma, N, M, \mathcal{E})$, nous définissons un automate $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$ (où Q est l'ensemble d'états, δ la fonction de transition et F l'ensemble des états finaux), qui reconnaît l'ensemble des arbres contenant une trace de $\mathscr{R}$.

L'automate $\mathcal{A}_{\mathscr{R}}$ est construit à partir des automates finis de mots associés aux expressions régulières apparaissant dans l'arbre $\mathscr{T}$: pour chaque expression régulière $\mathcal{E}_e$ (où $e = (w, w')$ est un arc de M), nous notons $\widetilde{L(\mathcal{E}_e)}$ le langage rationnel formé des mots miroirs de mots de $L(\mathcal{E}_e)$. L'idée intuitive est de construire $\mathcal{A}_{\mathscr{R}}$ de telle sorte que, étant donné un arbre $\mathcal{T}$ et un plongement p de $\mathscr{R}$ dans $\mathcal{T}$ (voir figure 8), un calcul *ascendant* de $\mathcal{A}_{\mathscr{R}}$ sur $\mathcal{T}$ réalise, sur la suite d'étiquettes $\widetilde{\sigma(e)} = \sigma_1\sigma_2...\sigma_{n-1}$ rencontrées lors du parcours *ascendant* du chemin P_e, une simulation d'un calcul d'un automate de mots reconnaissant le langage $\widetilde{L(\mathcal{E}_e)}$.
Pour chaque arc $e = (w, w')$ de M, on suppose donné un automate de mots finis $\mathcal{A}_e = (\Sigma, Q_e, \delta_e, t_e^0, f_e)$ reconnaissant $\widetilde{L(\mathcal{E}_e)}$. Sans perte de généralité, on suppose également safistaites les trois propriétés suivantes :

(i) les ensembles $\{Q_e, e \in M\}$ sont disjoints deux à deux

(ii) $\mathcal{A}_e$ a un unique état initial t_e^0 et un unique état final f_e

(iii) l'ensemble R_e des états de Q_e accessibles à partir de t_e^0 par une seule transition de δ_e ne sont accessibles à partir d'aucun autre état de Q_e.

Remarquons que le principe de simulation des automates $\mathcal{A}_e$ par $\mathcal{A}_{\mathscr{R}}$ et la propriété (iii) satisfaite par chacun de ces automates, impliquent que les nœuds $p(w')$, images par p du sommet w' d'un arc $e = (w, w')$ de M, sont les seuls nœuds associés à un état de R_e, par au moins un calcul réussi de $\mathcal{A}_{\mathscr{R}}$. Ainsi, les nœuds de $\mathcal{T}$ associés par p à un nœud de sélection de $\mathscr{R}$ (*i. e.* à un nœud de $\mathcal{N}(\vec{s})$) sont les seuls nœuds associés, par un calcul réussi de $\mathcal{A}_{\mathscr{R}}$, à un état de l'ensemble $Select(\mathcal{A}_{\mathscr{R}}) = \bigcup_{e=(w,w') \in M / w' \in \mathcal{N}(\vec{s})} R_e$.

Nous donnons en section 5.1.1 la construction formelle de $\mathcal{A}_{\mathscr{R}}$.

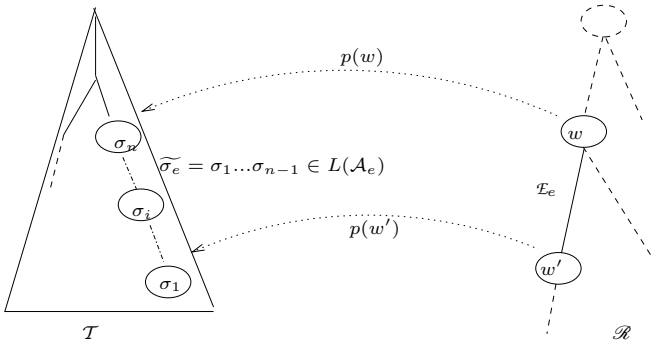


Figure 8. *Principe de construction de $\mathcal{A}_{\mathscr{R}}$*

5.1.1. *Construction de $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$*

Nous notons par τ la trace de $\mathscr{R}$ dans $\mathcal{T}$ qui doit être identifiée par un calcul réussi de $\mathcal{A}_{\mathscr{R}}$, et p son plongement associé.
L'ensemble des états est défini par : $Q = \cup_{e \in M} Q_e \cup \{f, g\}$ où f et g sont deux états n'apparaissant pas dans $\cup_{e \in M} Q_e$. L'état f joue le rôle d'état final, *i. e.* $F = \{f\}$, et l'état g est un état générique qui sera associé aux nœuds n'appartenant pas à la trace τ de $\mathscr{R}$ en cours de reconnaissance.
L'ensemble des transitions δ est donné en définissant ci-après, pour chaque étiquette x de Σ et état t de Q, l'ensemble $L(x,t)$ = $\{w \in Q^* / (x,t,w) \in \delta\}$:

– $L(x,g) = g^*$. Cet ensemble de transitions permet à l'automate $\mathcal{A}_{\mathscr{R}}$ d'associer l'état générique g, à un nœud w et à ses fils, dans le cas où w n'appartient pas à la trace τ en cours de reconnaissance (Transition 1 de la figure 9).

– $L(/, f) = g^* f_{e_1} g^*$ si $Out(\varepsilon) = \{e_1\}$ dans $\mathscr{T}$, $L(x,f) = \emptyset$ pour tout $x \neq' /'$. Cet ensemble de transitions permet l'arrêt d'un calcul réussi de $\mathcal{A}_{\mathscr{R}}$.

– Si $t \in Q_e$ pour un arc e de M, $L(x,t)$ est l'union de trois ensembles de transitions, $L(x,t) = L_1(x,t) \cup L_2(x,t) \cup L_3(x,t)$, définis comme suit :

- L'ensemble $L_1(x,t)$ n'est non vide que si $e = (w, w')$ avec w' feuille de $\mathscr{T}$. Cet ensemble de transitions permet à $\mathcal{A}_{\mathscr{R}}$ de démarrer un calcul de $\mathcal{A}_e$, à partir de l'image par p d'une feuille de $\mathscr{T}$, afin de reconnaître le mot miroir $\widetilde{\sigma(e)}$ (Transition 2 de la figure 9).

$$L_1(x,t) = \begin{cases} g^* & \text{si } w' \text{ est un nœud feuille de } \mathscr{T} \text{ et (x, } t_e^0\text{, t)} \in \delta_e \\ \emptyset & \text{sinon} \end{cases}$$

- L'ensemble $L_2(x,t)$ permet à $\mathcal{A}_{\mathscr{R}}$ de continuer un calcul de $\mathcal{A}_e$ dèja commencé (Transition 3 de la figure 9) :

$$L_2(x,t) = \bigcup_{\{t' / (x,t',t) \in \delta_e\}} g^* t' g^*$$

- L'ensemble $L_3(x,t)$ permet à $\mathcal{A}_{\mathscr{R}}$, si $e = (w,w')$ et $Out(w') = \{e_1, e_2, ..., e_k\}$, de démarrer un calcul de $\mathcal{A}_e$ à partir du nœud $p(w')$, afin de reconnaître le mot $\widetilde{\sigma(e)}$, lorsque les mots $\widetilde{\sigma(e_1)}, ..., \widetilde{\sigma(e_k)}$ ont déjà été reconnus (Transition 4 de la figure 9) :

$$L_3(x,t) = \begin{cases} g^* f_{e_1} g^* f_{e_2} g^* ... g^* f_{e_k} g^* & \text{si } w' \text{ n'est pas une feuille de } \mathscr{R} \\ & \text{et } (x, t_e^0, t) \in \delta_e, \\ \emptyset & \text{sinon} \end{cases}$$

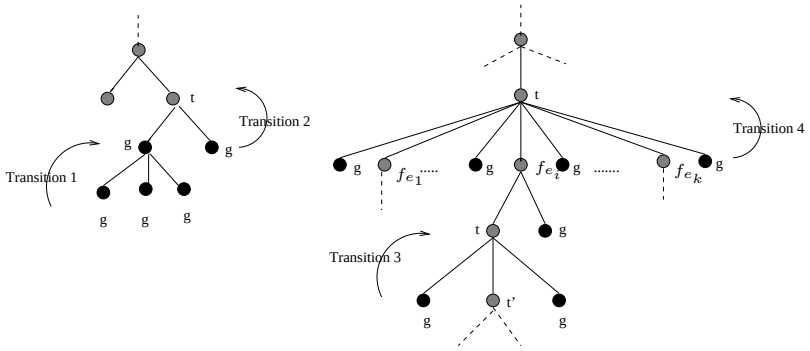


Figure 9. *Transitions*

5.1.2. *Construction de* $\bar{\mathcal{A}}_{\mathscr{R}}$=$(\Sigma, U, \eta, F)$

Nous modifions maintenant la construction de l'automate $\mathcal{A}_{\mathscr{R}}$ afin d'obtenir un nouvel automate $\bar{\mathcal{A}}_{\mathscr{R}}$ reconnaissant le même langage que $\mathcal{A}_{\mathscr{R}}$ mais permettant d'identifier tous les descendants de nœuds images par p des nœuds de selection de $\mathscr{R}$. L'idée est de changer légérement $\mathcal{A}_{\mathscr{R}}$ pour que les calculs réussis associent des états barrés à de tels nœuds.

Formellement, notons $\bar{Q}$ un ensemble de copies barrées des états de Q, $\overline{w}$ le mot $\bar{q_1}...\bar{q_k}$ si w est le mot $q_1...q_k$ de Q^*, et $\overline{S}$ l'ensemble $\{\bar{w}/w \in S\}$ si $S \subseteq Q^*$. On pose $\bar{\mathcal{A}}_{\mathscr{R}} = (\Sigma, U, \eta, F)$ avec $U = Q \cup \bar{Q}$, $F = \{f\}$ et on definit les ensembles de transitions associés à η comme suit :

- $L(x, \bar{g}) = \bar{g}^*$
- $L(x, \bar{t}) = \overline{L_1(x,t)} \cup \overline{L_2(x,t)} \cup \overline{L_3(x,t)}$
- $L(x,t) = \overline{L_1(x,t)} \cup \overline{L_3(x,t)}$ si $t \in R_e$, $e = (w,w')$ et $w' \in \mathcal{N}(\vec{s})$
- $L(x,g) = g^*$
- $L(x,t) = L_1(x,t) \cup L_2(x,t) \cup L_3(x,t)$ si $t \in Q_e$, $e = (w,w')$ et $w' \notin \mathcal{N}(\vec{s})$

Les deux premiers ensembles de transitions associent des états barrés à tous les nœuds descendants d'un nœud de sélection, tandis que le troisième permet à un calcul *ascendant* de $\bar{\mathcal{A}}_{\mathscr{R}}$ de passer d'états barrés à des états non barrés dès qu'un nœud de sélection est atteint.

Ainsi, dans tout calcul de $\bar{\mathcal{A}}_{\mathscr{R}}$ sur $\mathcal{T}$, identifiant la trace $\texttt{trace}_p(\mathscr{R}, \mathcal{T})$ d'un plongement p de $\mathscr{R}$ dans $\mathcal{T}$, les nœuds apparaissant sur cette trace ou descendant

d'un nœud de sélection (c'est-à-dire les nœuds de $\mathcal{N}(\texttt{trace}_p(\mathcal{V},\mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$) sont caractérisés par le fait qu'ils sont associés à un état de $U \setminus \{g\}$, c'est-à-dire à des états barrés ou à des états de $\cup_{e\in M} Q_e \cup \{f\}$. Cette propriété sera utilisée plus loin.

5.2. *Construction de* $\mathcal{A}$

L'automate d'arbres $\mathcal{A}$ reconnaissant le langage $\mathcal{L}$, qui intervient dans le critère d'indépendance, est construit à partir des automates $\mathcal{A}_\mathcal{C} = (\Sigma, Q_\mathcal{C}, \delta_\mathcal{C}, F_\mathcal{C})$ et $\bar{\mathcal{A}}_\mathcal{V} = (\Sigma, U_\mathcal{V}, \eta_\mathcal{V}, F_\mathcal{V})$ obtenus, pour $\mathcal{A}_\mathcal{C}$ à partir de la classe de mises à jour $\mathcal{C}$ en utilisant la construction de la section 5.1.1 et, pour $\bar{\mathcal{A}}_\mathcal{V}$ à partir de la requête de vue $\mathcal{V}$ en utilisant la construction de la section 5.1.2. La construction de $\mathcal{A}$ à partir de $\mathcal{A}_\mathcal{C}$ et $\bar{\mathcal{A}}_\mathcal{V}$, met en œuvre deux constructeurs classiques d'automates que nous présentons ci-après.

Automate produit $\mathcal{A}_1 \times \mathcal{A}_2$
Soit $\mathcal{A}_1$=$(\Sigma, Q_1, \delta_1, F_1 \subseteq Q_1)$ et $\mathcal{A}_2$=$(\Sigma, Q_2, \delta_2, F_2 \subseteq Q_2)$ deux automates d'arbres. Le langage $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ est reconnu par l'automate produit $\mathcal{A}_1 \times \mathcal{A}_2$=$(\Sigma, Q_1 \times Q_2, \Delta, F_1 \times F_2)$ où Δ est donné par : $(x,(q,q'),(q_1,q'_1)(q_2,q'_2)\ldots(q_n,q'_n)) \in \Delta$ si et seulement si $(x,q,q_1q_2\ldots q_n) \in \delta_1$ et $(x,q',q'_1q'_2\ldots q'_n) \in \delta_2$

Automate à états sélectifs $\sigma(\mathcal{B}, S)$
Soit $\mathcal{B} = (\Sigma, U, \eta, F_U \subseteq U)$ un automate d'arbres et $S \subseteq U$ un sous-ensemble d'états de U. L'ensemble des arbres $\mathcal{T}$ pour lesquels il existe un calcul réussi de $\mathcal{B}$ attribuant un état de S à au moins un nœud de $\mathcal{T}$ est un langage rationnel d'arbres reconnu par l'automate $\sigma(\mathcal{B}, S)$, déduit de $\mathcal{B}$ comme suit : le fonctionnement de $\sigma(\mathcal{B}, S)$ est semblable à celui de $\mathcal{B}$ mais utilise un ensemble supplémentaire $\widehat{U}$ (copie de U) d'états chapeautés ; les calculs de $\sigma(\mathcal{B}, S)$ sur un arbre $\mathcal{T}$ sont identiques à ceux de $\mathcal{B}$ sur $\mathcal{T}$, mais attribuent des états chapeautés aux ascendants des nœuds ayant été associés à un état de S. Les états finaux de $\sigma(\mathcal{B}, S)$ sont les états finaux chapeautés de $\mathcal{B}$, ce qui assure que tout calcul réussi de $\sigma(\mathcal{B}, S)$ correspond à un calcul réussi de $\mathcal{B}$ utilisant au moins un état de S. Formellement, si π est l'application de $U \cup \hat{U}$ dans U définie $\forall u \in U$ par $\pi(\hat{u}) = \pi(u) = u$, on pose : $\sigma(\mathcal{B}, S) = (\Sigma, U \cup \hat{U}, \eta \cup \gamma, \hat{F_U})$ avec $\hat{F_U} = \{\hat{u}/u \in F_U\}$ et $\gamma = \{(x,\hat{u},w)/w \in (U \cup \hat{U})^* S (U \cup \hat{U})^* et (x,u,\pi(w)) \in \eta\}$ $\cup \{(x,\hat{u},w)/w \in (U \cup \hat{U})^* \hat{U} (U \cup \hat{U})^* et (x,u,\pi(w)) \in \eta\}$.
A partir de maintenant, nous supposons qu'un schéma $\mathcal{S}c$ est disponible et spécifié par un automate fini d'arbre $\mathcal{A}_{\mathcal{S}c}$, c'est-à-dire : $L(\mathcal{A}_{\mathcal{S}c}) = \texttt{valide}(\mathcal{S}c)$.

Proposition 5. *Le langage $\mathcal{L}$ est reconnu par l'automate d'arbres $\mathcal{A}$ défini par :* $\mathcal{A} = \mathcal{A}_{\mathcal{S}c} \times \sigma(\mathcal{A}_\mathcal{C} \times \bar{\mathcal{A}}_\mathcal{V}, S)$ *où* $S = R_\mathcal{C} \times (U_\mathcal{V} \setminus \{g\})$.

Démonstration. Rappelons que le langage $\mathcal{L}$ est l'ensemble des arbres $\mathcal{T}$ vérifiant :
(i) $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$
(ii) il existe une trace $\texttt{trace}_p(\mathcal{V},\mathcal{T})$ de $\mathcal{V}$ dans $\mathcal{T}$ selon un plongement p et il existe une trace $\texttt{trace}_{p'}(\mathcal{C},\mathcal{T})$ de $\mathcal{C}$ dans $\mathcal{T}$ selon un plongement p' telles que :
$p'(s_\mathcal{C}) \in \mathcal{N}(\texttt{trace}_p(\mathcal{V},\mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$
$\mathcal{A}_\mathcal{C} \times \bar{\mathcal{A}}_\mathcal{V}$ reconnait l'ensemble des arbres contenant une trace de $\mathcal{V}$ et une trace de $\mathcal{C}$. Par ailleurs un calcul réussi de $\mathcal{A}_\mathcal{C} \times \bar{\mathcal{A}}_\mathcal{V}$ sur un arbre $\mathcal{T}$ attribue aux nœuds de

$p'(s_{\mathcal{C}}) \cap (\mathcal{N}(\texttt{trace}_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T})))$ un état de $Select(\mathcal{A}_{\mathcal{C}}) \times (U_{\mathcal{V}} \setminus \{g\})$. Ainsi l'ensemble des arbres vérifiant la condition (ii) est reconnu par l'automate $\sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}_{\mathcal{V}}}, S)$ où $S = Select(\mathcal{A}_{\mathcal{C}}) \times (U_{\mathcal{V}} \setminus \{g\})$. L'ajout de la condition (i) assure l'égalité $\mathcal{L} = L(\mathcal{A})$.

5.3. *Etude de la Complexité*

Dans cette section nous étudions la complexité de la construction de l'automate $\mathcal{A} = \mathcal{A}_{Sc} \times \sigma(\mathcal{A}_{\mathcal{C}} \times \bar{\mathcal{A}_{\mathcal{V}}}, S)$. Nous commençons d'abord par la complexité de la construction de l'automate reconnaîssant une trace.

Lemme 2. *Soit $\mathcal{A}_{\mathscr{R}} = (\Sigma, Q, \delta, F)$ l'automate construit en section 5.1.1 à partir de la requête $\mathscr{R}$=$(\mathscr{T}, \vec{s})$ avec $\mathscr{T}$=$(\Sigma, N, M, \mathcal{E})$. Si a_m désigne l'arité maximale des nœuds de N, la taille $|\mathcal{A}_{\mathscr{R}}|$ de $\mathscr{R}$ est en $O(|\Sigma| \times |\mathscr{R}| \times a_m)$.*

Démonstration. Soit $t \in Q$, $x \in \Sigma$ et $\mathcal{A}_{L(x,t)}$ un automate de mots reconnaissant le langage $L(x,t)$ = $\{w \in Q^* / (x,t,w) \in \delta\}$. Conformément à la définition de $\mathcal{A}_{\mathscr{R}}$ on a :

$$\begin{aligned} |\mathcal{A}_{\mathscr{R}}| &= |Q| + \sum_{(x,t)\in\Sigma\times Q} |\mathcal{A}_{L(x,t)}| \\ &= |Q| + |\mathcal{A}_{L(/,f)}| + \sum_{x\in\Sigma} (|\mathcal{A}_{L(x,g)}|) \\ &\quad + \sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_1(x,t)}| + |\mathcal{A}_{L_2(x,t)}| + |\mathcal{A}_{L_3(x,t)}|) \end{aligned}$$

où Q_M représente $\cup_{e\in M} Q_e$.

Rappelons tout d'abord que $|\mathscr{R}| = |N| + \sum_{e\in M} |\mathcal{A}_e|$ où $\mathcal{A}_e$ est l'automate associé à l'expression régulière $\mathcal{E}_e$. Nous montrons dans la suite que chaque terme de la somme exprimant $|\mathcal{A}_{\mathscr{R}}|$ est en $O(|\Sigma| \times |\mathscr{R}| \times a_m)$:

– $|Q|$ = $|\bigcup_{e\in M} Q_e \cup \{f, g\}|$ est en $O(|\mathscr{R}|)$

– $\forall e \in M, \forall (x,t) \in \Sigma \times Q_e, |\mathcal{A}_{L(x,g)}|$ et $|\mathcal{A}_{L_1(x,t)}|$ sont en $O(1)$

Ainsi $\sum_{x\in\Sigma} (|\mathcal{A}_{L(x,g)}|)$ est en $O(|\Sigma|)$ et $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_1(x,t)}|)$ est en $O(|\Sigma| \times |\mathscr{R}|)$

– Il existe une constante K telle que :

$\forall e \in M, \forall (x,t) \in \Sigma \times Q_e, |\mathcal{A}_{L_2(x,t)}| \leqslant K \times |\{t' / (x,t',t) \in \delta_e\}|$

De l'égalité $\sum_{(x,t)\in\Sigma\times Q_M} (|\mathcal{A}_{L_2(x,t)}|)$ = $\sum_{e\in M} (\sum_{(x,t)\in\Sigma\times Q_e} (|\mathcal{A}_{L_2(x,t)}|))$, on déduit que

$\sum_{(x,t)\in\Sigma\times Q_M}(|\mathcal{A}_{L_2(x,t)}|)$ est en $O(\sum_{e\in M}|\delta_e|)$.
Or $\sum_{e\in M}|\delta_e| \leqslant |\mathscr{R}|$, et donc $\sum_{(x,t)\in\Sigma\times Q_M}(|\mathcal{A}_{L_2(x,t)}|)$ est en $O(|\mathscr{R}|)$.

– $\forall e \in M, \forall (x,t) \in \Sigma \times Q_e, |\mathcal{A}_{L_3(x,t)}|$ est en $O(a_m)$ et il y a au plus $\sum_{e\in M}|\delta_e|$ couples (x,t) pour lesquels $L_3(x,t)$ est non vide.
Ainsi $\sum_{(x,t)\in\Sigma\times Q_M}(|\mathcal{A}_{L_3(x,t)}|)$ est en $O(a_m \times |\mathscr{R}|)$.

– $|\mathcal{A}_{L(/,f)}|$ est en $O(1)$

Nous étudions maintenant la taille de l'automate avec états sélectifs utilisé dans la construction de $\mathcal{A}$.

Lemme 3. *Soit $\mathcal{B} = (\Sigma, U, \eta, F_U \subseteq U)$ un automate d'arbres, $S \subseteq U$ un sous-ensemble d'états de U et $\Im = \sigma(\mathcal{B}, S)$ l'automate avec états sélectifs défini en section 5.2. La taille $|\Im|$ de $\Im$ est en $O(|\mathcal{B}|)$.*

Démonstration. Nous avons :
$|\Im| = 2|U| + \sum_{(x,\hat{u})\in\Sigma\times\hat{U}}|\mathcal{A}_{L_\Im(x,\hat{u})}| + \sum_{(x,u)\in\Sigma\times U}|\mathcal{A}_{L_\Im(x,u)}|$.
Pour tout $(x,u) \in \Sigma \times U$, $L_\Im(x,u) = L_\mathcal{B}(x,u)$.
Par ailleurs, $L_\Im(x,\hat{u}) = \pi^{-1}(L_\mathcal{B}(x,u)) \cap (U \cup \hat{U})^*(\hat{U} \cup S)(U \cup \hat{U})^*$, et donc $|\mathcal{A}_{L_\Im(x,\hat{u})}|$ est en $O(|\mathcal{A}_{L_\mathcal{B}(x,u)}|)$.
Du fait que $|\mathcal{B}| = |U| + \sum_{(x,u)\in\Sigma\times U}|\mathcal{A}_{L_\mathcal{B}(x,u)}|$, on déduit que $|\Im|$ est en $O(|\mathcal{B}|)$.

Nous pouvons maintenant combiner les résultats des Lemmes 1 et 2.

Proposition 6. *La taille $|\mathcal{A}|$ de l'automate $\mathcal{A} = \mathcal{A}_{\mathcal{S}c} \times \sigma(\mathcal{A}_\mathcal{C} \times \bar{\mathcal{A}_\mathcal{V}}, S)$ est en $O(a_\mathcal{C} a_\mathcal{V} \times |\Sigma|^2 \times |\mathcal{A}_{\mathcal{S}c}| \times |\mathcal{C}| \times |\mathcal{V}|)$, où $a_\mathcal{C}$ et $a_\mathcal{V}$ sont les arités maximales des nœuds de $\mathcal{C}$ et $\mathcal{V}$ respectivement.*

Démonstration. Nous avons $|\mathcal{A}| \leqslant |\mathcal{A}_{\mathcal{S}c}| \times |\sigma(\mathcal{A}_\mathcal{C} \times \bar{\mathcal{A}_\mathcal{V}}, S)|$ et on déduit facilement de la construction donnée en section 5.1.2 que $|\bar{\mathcal{A}_\mathcal{V}}| \leq 2|\mathcal{A}_\mathcal{V}|$. Le résultat découle alors des lemmes 1 et 2.

Proposition 7. *La complexité en temps du test de vacuité du langage $\mathcal{L}$ est en $O(a_\mathcal{C}^2\ a_\mathcal{V}^2\ |\Sigma|^4 |\mathcal{A}_{\mathcal{S}c}|^2 \times |\mathcal{C}|^2 \times |\mathcal{V}|^2)$. Ainsi le critère d'indépendance est décidable en temps polynomial.*

Démonstration. Comme $\mathcal{L} = L(\mathcal{A})$, il suffit d'utiliser l'algorithme classique de test de vacuité de $L(\mathcal{A})$ qui calcule, par saturation jusqu'à obtention d'un point fixe, le sous-ensemble des états accessibles de $\mathcal{A}$. La complexité en temps de cet algorithme est polynomiale en O($|\mathcal{A}|^2$). Le résultat découle alors de la Proposition 6.

6. Conclusion

Dans ce travail nous avons étudié le problème d'indépendance entre vues et mises à jour. Notre principale contribution est d'avoir exhibé une condition suffisante d'indépendance entre une requête de vue et une classe de mises à jour, vérifiable en temps polynomial. Cette condition est nécessaire et suffisante dans le cas particulier de requêtes de vue linéaires. Nous avons également montré que ce problème d'indépendance est en général PSPACE-difficile.

Pour cette étude, nous avons choisi de spécifier les requêtes de vue et les classes de mises à jour par des requêtes arbres régulières. Ce langage formel, incomparable avec XPath, permet cependant d'exprimer les requêtes $CoreXPath^{+}$. Nos résultats peuvent ainsi être appliqués lorsque vues et mises à jour sont exprimées par des requêtes de $CoreXPath^{+}$. Une implantation du test d'indépendance ainsi qu'une étude expérimentale reste à faire, en particulier pour mesurer le gain obtenu par une exécution du test d'indépendance à la place d'une réévaluation systématique de la vue aprés la mise à jour.

Notre analyse d'indépendance entre requêtes de vue et classes de mises à jour peut être également vue comme une analyse d'indépendance entre deux requêtes RAR et peut être utilisée dans d'autres contextes d'applications : le problème de la commutativité entre deux requêtes de mises à jour étudié dans [GHE 07] et dans [BEN 05] en est un exemple. Ainsi nous pensons que notre approche est suffisamment générale et adaptable à d'autres contextes d'applications nécessitant une analyse de dépendances entre requêtes RAR.

Remerciements

Ce travail a bénéficié d'une aide de l'Agence Nationale de la Recherche portant la référence ANR-08-DEFIS-04.

7. Bibliographie

[ABI 98] ABITEBOUL S., MCHUGH J., RYS M., VASSALOS V., WIENER J. L., « Incremental Maintenance for Materialized Views over Semistructured Data », *VLDB '98 : Proceedings of the 24rd International Conference on Very Large Data Bases*, San Francisco, CA, USA, 1998, p. 38–49.

[ALI 03] ALI M. A., FERNANDES A. A., PATON N. W., « MOVIE : An incremental maintenance system for materialized object views », *Data & Knowledge Engineering*, vol. 47, 2003, p. 131–166, Elsevier.

[ARI 07] ARION A., BENZAKEN V., MANOLESCU I., PAPAKONSTANTINOU Y., « Structured Materialized Views for XML Queries », *VLDB '07 : Proceedings of the 33rd international conference on Very large data bases*, Vienna, Austria, 2007, p. 87–98.

[BAL 04] BALMIN A., ÖZCAN F., BEYER K. S., COCHRANE R. J., PIRAHESH H., « A framework for using materialized XPath views in XML query processing », *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, Toronto, Canada, 2004.

[BEN 05] BENEDIKT M., BONIFATI A., FLESCA S., VYAS A., « Verification of Tree Updates for Optimization. », *CAV*, vol. 3576, Springer, 2005, p. 379–393.

[BLA 86] BLAKELEY J. A., LARSON P.-A., TOMPA F. W., « Efficiently Updating Materialized Views », *SIGMOD Conference*, 1986, p. 61–71.

[BLA 89] BLAKELEY J. A., COBURN N., LARSON P.-A., « Updating Derived Relations : Detecting Irrelevant and Autonomously Computable Updates », *ACM Transactions on Database Systems*, vol. 14, n° 3, 1989, p. 369–400.

[GHE 07] GHELLI G., ROSE K. H., SIMÉON J., « Commutativity Analysis in XML Update Languages », *ICDT*, 2007, p. 374–388.

[GRI 95] GRIFFIN T., LIBKIN L., « Incremental maintenance of views with duplicates », *SIGMOD Rec.*, vol. 24, n° 2, 1995, p. 328–339, ACM.

[GUP 93] GUPTA A., MUMICK I. S., SUBRAHMANIAN V. S., « Maintaining views incrementally », *ACM SIGMOD international conference on Management of data*, ACM, 1993, p. 157–166.

[GUP 99] GUPTA A., MUMICK I. S., « Maintenance of materialized views : problems, techniques, and applications », *Materialized views : techniques, implementations, and applications*, Cambridge, MA, USA, 1999, MIT Press, p. 145–157.

[LAK 06] LAKSHMANAN L. V. S., WANG H., ZHAO Z., « Answering tree pattern queries using views », *VLDB '06 : Proceedings of the 32nd international conference on Very large data bases*, Seoul, Korea, 2006, p. 571–582.

[LAU 01] LAURENT D., LECHTENBÖRGER J., SPYRATOS N., VOSSEN G., « Monotonic complements for independent data warehouses », *The VLDB Journal*, vol. 10, n° 4, 2001, p. 295–315.

[LIE 00] LIEFKE H., DAVIDSON S. B., « View Maintenance for Hierarchical Semistructured Data », *Data Warehousing and Knowledge Discovery*, 2000, p. 114–125.

[MAR 04] MARTENS W., NEVEN F., SCHWENTICK T., CENTRUM L. U., « Complexity of decision problems for simple regular expressions », *In Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, Springer, 2004, p. 889–900.

[MIK 04] MIKLAU G., SUCIU D., « Containment and equivalence for a fragment of XPath », *Journal of the ACM*, vol. 51, n° 1, 2004, p. 2–45, ACM.

[ONI 05] ONIZUKA M., CHAN F. Y., MICHIGAMI R., HONISHI T., « Incremental Maintenance for Materialized XPath/XSLT Views », *WWW 2005*, Chiba Japan, 2005, ACM.

[QUA 01] QUAN L., CHEN L., RUNDENSTEINER E. A., « Argos : Efficient Refresh in an XQL-Based Web Caching System », 2001, Lecture Notes in Computer Science.

[RAG 06] RAGHAVACHARI M., SHMUELI O., « Conflicting XML Updates », *Advances in Database Technology - EDBT*, vol. 3896, 2006, p. 552–569.

[SAW 05] SAWIRES A., TATEMURA J., PO O., AGRAWAL D., CANDAN K. S., « Incremental maintenance of path-expression views », *ACM SIGMOD international conference on Management of data*, 2005, p. 443–454.

[SCH 91] SCHOLL M. H., LAASCH C., TRESCH M., « Updatable Views in Object-Oriented Databases », *Proc. 2nd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, 1991.

[SEG 05] SEGOUFIN L., VIANU V., « Views and Queries : Determinacy and Rewriting », *Symposium on Principles of Database Systems (PODS)*, 2005.

[VIS 96] VISTA D., « Optimizing Incremental View Maintenance Expressions In Relational Databases », PHD, University of Toronto, 1996.

[ZHU 98] ZHUGE Y., GARCIA-MOLINA H., « Graph Structured Views and Their Incremental Maintenance », *Proc. 14th IEEE Conf. Data Engineering, ICDE*, IEEE Computer Society, 1998, p. 116–125.

8. Appendice

Lemme 1 ***(Stabilité des unions de requêtes RAR par extension et filtrage)***

Notation : si $\tau_{i,j}$ désigne la transposition des éléments d'indices i et j d'un n-uplet, et si $\mathscr{R} = (\mathscr{T}, \vec{s})$ est une requête RAR, on note $\tau_{i,j}(\mathscr{R})$ la requête RAR $\mathscr{R} = (\mathscr{T}, \tau_{i,j}(\vec{s}))$.

La preuve du Lemme 1 utilise les propriétés suivantes des opérations d'extension et de filtrage de requêtes RAR, qui s'établissent sans difficulté : si f,g sont des filtres de $CoreXPath^+$, f_i $(i = 1, ..., k)$ des filtres unaires de $CoreXPath^+$, $\mathscr{R} = (\mathscr{T}, \vec{s})$ et $\mathscr{R}_i$ $(i = 1, ..., k)$ des requêtes RAR n-aires, λ une lettre de Σ et $axis$ un axe navigationnel de $XPath$:

1) $\mathscr{R}[f] = \Pi_{(1,2,...,n)}(\mathscr{R}.f)$

2) $\mathscr{R}.(/g) = \Pi_{(1,2,...,n,n+2)}(\mathscr{R}_\epsilon.g)$ avec $\mathscr{R}_\epsilon = (\mathscr{T}, (\vec{s}, \epsilon))$

3) $(\cup_{i=1}^{k}\mathscr{R}_i).f = \cup_{i=1}^{k}(\mathscr{R}_i.f)$

4) $\mathscr{R}.(axis :: \lambda[f]) = \Pi_{(1,...,n+1)}((\mathscr{R}.axis :: \lambda).f)$

5) $\mathscr{R}.((f_1\ and\ ...and f_k)\ and\ f_{k+1}) = \tau_{n,n+k}(\tau_{n,n+k}(\mathscr{R}.(f_1\ and...and f_k)).f_{k+1})$

Notons que : dans la propriété 2, la requête $\mathscr{R}_\epsilon$ est introduite afin de tenir compte du caractère absolu du filtre $/g$; dans la propriété 5, la transposition permet d'échanger les nœuds d'indices n et $n+k$ afin que l'extension par le filtre f_{k+1} soit correctement réalisée à partir du n^{ieme} nœud de sélection de la requête $\mathscr{R}$.

La stabilité des unions de requêtes RAR par extension se démontre alors par induction sur la longueur du filtre f. Nous esquissons ici les grandes lignes du raisonnement :
(i) les cas de base ($f = axis :: \lambda$ et $f = /axis :: \lambda$) sont traités en construisant, pour chaque valeur possible de l'axe navigationnel $axis$, un arbre spécifique déduit de l'arbre $\mathscr{T}$ de $\mathscr{R}$, permettant d'exprimer, sous forme d'une union de requêtes RAR monadiques, la requête $\mathscr{R}.f$
(ii) l'étape d'induction utilise les propriétés de l'opération d'extension :
- Si f est de la forme $axis :: \lambda[g]$, on utilise la propriété 4, le fait que $\mathscr{R}.(axis :: \lambda)$ est exprimable par une union de requêtes RAR monadiques, la propriété 3 et l'hypothèse d'induction.
- Si f est de la forme $/g$, on utilise la propriété 2 et l'hypothèse d'induction.
- Si f est de la forme $f_1\ and\ ...f_k$, avec f_i unaire $(i = 1, ..., k)$ on raisonne par récurrence sur k et on utilise la propriété 5.

La stabilité des unions de requêtes RAR par filtrage est une conséquence directe de la propriété 1 et de la stabilité des unions de requêtes RAR par extension.